VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Automatická kompozice služeb

Automatic Composition of Services

2018 David Katanik

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. David Katanik**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Automatická kompozice služeb
Automatic Composition of Services

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je prozkoumat současný stav v oblasti automatické kompozice služeb, navrhnout a vytvořit prototyp platformy, která bude demonstrovat vybrané možnosti. Nejvíce pozornosti bude věnováno webovým službám respektive "RESTful" webovým službám.

Práce bude zejména obsahovat:
1. Popis současného stavu dané problematiky.
2. Model navrhnuté architektury vytvářené platformy.
3. Implementaci prototypu platformy, která bude dostupná v repozitáři aplikace git.cs.vsb.cz. Vše bude sestavitelné pomocí systému maven či gradle a automaticky nasaditelné např. pomocí nástroje salt.
4. "Skripty" pro automatizované testování a výsledky experimentů.
5. Zhodnocení řešení a popis návrhu dalšího rozvoje.

Seznam doporučené odborné literatury:

[1] FANJIANG, Y. Y., Y. SYU, S. P. MA a J. Y. KUO, 2017. An Overview and Classification of Service Description Approaches in Automated Service Composition Research. IEEE Transactions on Services Computing [online]. 10(2), 176–189. ISSN 1939-1374. Dostupné z: doi:10.1109/TSC.2015.2461538
[2] KUMARA, I., J. HAN, A. COLMAN a M. KAPURUGE, 2017. Software-Defined Service Networking: Performance Differentiation in Shared Multi-Tenant Cloud Applications. IEEE Transactions on Services Computing [online]. 10(1), 9–22. ISSN 1939-1374. Dostupné z: doi:10.1109/TSC.2016.2594061

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Kožusznik, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 23. dubna 2018 .....

Velice rád bych poděkoval Ing. Janu Kožusznikovi, Ph.D. za příkladné vedení práce, věcné připomínky, spolupráci, motivaci a umožnění vypracování tématu.

Abstrakt

Cílem této diplomové práce je prozkoumat současný stav automatické kompozice služeb. Práce se výhradně zaměřuje na kompozici RESTful webových služeb a popisuje nejnovější trendy problematiky spolu s dalším předpokládaným vývojem. Dalším cílem práce je vytvoření prototypového řešení využívající určitou formu automatické kompozice. Implementované prototypové řešení využívá poloautomatickou kompozici RESTful webových služeb založených na microservices architektuře. Kompozice využívá podnikový BPMN proces k definování úkonů orchestrace. Řešení prokazuje správnost konceptu poloautomatické kompozice služeb a potvrzuje předpoklad dalšího vývoje a integrace konceptu do produkčních řešení.

Klíčová slova: Kompozice webových služeb, RESTful webové služby, Java, Spring framework, BPMN, Camunda, Docker

Abstract

The aim of this diploma thesis is to examine the current state of the Automatic Service Composition. The work focuses exclusively on the composition of RESTful Web Services and describes the latest trends in this area along with other anticipated developments. Another goal of this thesis is to create a prototype solution using a certain form of an Automatic Composition. The implemented prototype solution uses the Semi-Automatic Composition of RESTful Web Services which are based on microservices architecture. The Composition uses the BPMN business process to define orchestration of tasks. The solution demonstrates the correctness of the concept of Semi-Automatic Service Composition and confirms the premise of further development and integration of concept into production solutions.

Key Words: Web Service Composition, RESTful Web Services, Java, Spring framework, BPMN, Camunda, Docker

Obsah

Seznam použitých zkratek a symbolů **8**

Seznam obrázků **10**

Seznam tabulek **11**

Seznam výpisů zdrojového kódu **12**

1 Úvod **13**

2 Webová služba **14**
2.1 Formální definice a popis 14
2.2 Základní funkční jednotky webových služeb využívající SOAP i REST 15
2.3 SOAP 17
2.4 REST 19
2.5 Servisně orientovaná architektura 24
2.6 Popis služby - Service description (SD) 25
2.7 Kvalita služby - QoS 27
2.8 Bezpečnost 29

3 Kompozice webových služeb **31**
3.1 Statická a dynamická kompozice 31
3.2 Manuální, poloautomatická a automatická kompozice 32
3.3 Příklad procesu objednání s náznakem webových služeb 33

4 Automatická kompozice webových služeb **34**
4.1 Motivace 34
4.2 Orchestrace 34
4.3 Choreografie 38
4.4 Sémantický web 39

5 Trend a předpoklad vývoje kompozitních služeb **44**
5.1 Zdokonalování webových služeb 44
5.2 Vylepšení anotace služeb, automatický popis 45
5.3 Umělá inteligence při kompozici založené na QoS 46
5.4 Využití Sémantického Webu 46
5.5 Cloudová orchestrace 46

6 Prototypové řešení **48**
6.1 Vize a požadavky na koncept 48
6.2 Model navržené architektury vytvářené platformy 49
6.3 Popis implementace 50
6.4 Výsledky experimentů a popis konceptu 57
6.5 Zhodnocení řešení a popis návrhu dalšího rozvoje konceptu 58

7 Závěr **59**

Literatura **60**

Přílohy **64**

A Seznam příloh na CD **65**

Seznam použitých zkratek a symbolů

AI	–	Artificial Intelligence
API	–	Application Programming Interface
B2B	–	Business-To-Business
BPMN	–	Business Process Model and Notation
CSS	–	Cascading Style Sheets
DMN	–	Decision Model and Notation
DoS	–	Denial of Service
E2E	–	End to End
GUI	–	Graphical User Interface
HATEOAS	–	Hypermedia as the Engine of Application State
HTML	–	HyperText Markup Language
HTTP	–	HyperText Transfer Protocol
IT	–	Informační Technologie
IRI	–	Internationalized Resource Identifier
JS	–	JavaScript
JSON	–	JavaScript Object Notation
JSON-LD	–	JavaScript Object Notation - Linked Data
JVM	–	Java Virtual Machine
MIME	–	Multipurpose Internet Mail Extensions
MITM	–	Man In The Middle
MVC	–	Model-View-Controller
N3	–	Notation3
OASIS	–	Organization for the Advancement of Structured Information Standards
OSS	–	Open Source Software
OWL	–	Web Ontology Language
POM	–	Project Object Model
PPP	–	Point to Point Protocol
QoS	–	Quality of Service
RDF	–	Resource Description Framework
RDFS	–	Resource Description Framework Schema
REST	–	Representational State Transfer
RIF	–	Rule Interchange Format
SaaS	–	Software as a Service
SD	–	Service Description
SKOS	–	Simple Knowledge Organization System

SOA	–	Service Oriented Architecture
SOAP	–	Simple Object Access Protocol
SPARQL	–	Simple Protocol and RDF Query Language
Turtle	–	Terse RDF Triple Language
UDDI	–	Universal Description, Discovery, and Integration
URI	–	Uniform Resource Identifier
WADL	–	Web Application Description Language
WSDL	–	Web Services Description Language
WS-BPEL	–	Web Services Business Process Execution Language
WS-CDL	–	Web Services Choreography Description Language
W3C	–	World Wide Web Consortium
XML	–	eXtensible Markup Language
XSD	–	XML Schema Definition
YAML	–	Ain't Markup Language

Seznam obrázků

1 Architektura webových služeb s technologií SOAP 15
2 SOAP model 17
3 SOA meta-model [8] 24
4 Využití SD 25
5 Architektura vyhodnocování kvality webové služby [13] 27
6 Potenciální využití webových služeb v částech procesu objednávky 33
7 Schéma orchestrace 34
8 Vztah BPMN a DMN [22] 36
9 Schéma choreografie [19] 38
10 Ilustrace architektury Sémantického Webu [26] 39
11 Struktura OWL 2 [34] 42
12 Trend Microservices od roku 2010 [39] 45
13 Trend REST API, SOA a Microservice od roku 2004 [40] 45
14 Trend Cloudové orchestrace od roku 2004 [45] 47
15 Model navržené architektury vytvářené platformy 49
16 Proces nasazený v prototypu 52
17 Proces rezervace cesty 53

Seznam tabulek

1 Nejběžnější faktory QoS a související subfaktory [13] 28
2 Výběr volně dostupných modelovacích produktů pro standard BPMN 2.0 aktivních od roku 2015 [23][24][25] 37

Seznam výpisů zdrojového kódu

1 Ukázka WSDL služby 16
2 Ukázka SOAP požadavku na službu 18
3 Ukázka SOAP odpovědi na požadavek 18
4 Ukázka WADL 23
5 Ukázka YAML souboru pro webovou službu 50
6 Ukázka REST API 51
7 Ukázka Docker souboru 53
8 Ukázka Docker compose souboru 54

1 Úvod

Webové služby patří v současnosti mezi jeden z nejskloňovanějších pojmů v IT. Webové služby (část 2) jsou součástí nejnovějších softwarových řešení a s rostoucím tempem využívání těchto prostředků bylo nutné definovat nový architektonický styl[1]. Servisně orientovaná architektura (část 2.5) nabízí unifikovaný přístup pro navrhování softwarů jako službu (SaaS), a tím se stala trendem. To způsobilo, že jsou na webové služby kladeny další požadavky v podobě QoS (část 2.7) a bezpečnosti (část 2.8).

Dalším rozšířeným pojmem v IT je REST (část 2.4), který poskytuje množinu architektonických omezení vedoucí ke zlepšení softwaru jako celku. Pakliže dojde ke spojení těchto dvou pojmů, vznikne REST webová služba. Která využívá veškeré kladné vlastnosti obou zmíněných architektur.

REST webová služba otevírá nové možnosti na poli vývoje softwarů. Tato práce se zaobírá momentálně nejzajímavější možností, a to automatickou (popř. poloautomatickou) kompozicí webových služeb (část 3.2). Kompozice je mechanismus kombinující více služeb, a to i těch, které již jsou v kompozici, k vytvoření komplexnějších funkcionalit, čímž zvyšuje jejich potenciál.

Automatickou kompozici lze pojmout dvěma hlavními způsoby, orchestrací (část 4.2) a choreografií (část 4.3). První způsob je v praxi využívanější, a proto se tato práce zaměřuje na orchestraci. Automatická kompozice vyžaduje znát sémantiku služeb (část 4.4), jelikož více služeb může mít stejné parametry, ale jinou funkcionalitu. Z tohoto důvodu se častěji používá poloautomatická kompozice prostřednictvím podnikového procesu. Spojení podnikového procesu s REST webovými službami je moderním přístupem kompozice webových služeb.

Dalším pokrokem je využití microservices architektury (část 5.1.3), což umožňuje separaci komplexních služeb na menší, zajišťující dynamičtější, jednodušší a rychlejší služby. Podnikový proces tak může být méně složitý a zahrnuje větší množství úkonů. Součástí této práce je prototyp (část 6) konceptu pro orchestraci microservices REST webových služeb.

2 Webová služba

Pojem webová služba začal být častěji zmiňován se vzrůstající rozšířeností internetu a velkých podnikových softwarů. Původním účelem bylo propojení jednotlivých softwarových aplikací (výhradně B2B), které pracují na rozdílných platformách. Webové služby jim poskytují standardní prostředky k interoperabilitě. S dalším rozšiřováním webových aplikací a digitalizací bylo zapotřebí rozšířit původní návrh, který se orientoval na specifickou strukturu využívající SOAP, o kompatibilitu s technologiemi jako je REST.

2.1 Formální definice a popis

Dle definice W3C je webová služba softwarový systém navržený pro podporu interakce mezi stroji v síti. Služba poskytuje rozhraní popsané ve strojově zpracovatelném formátu (WSDL). Ostatní systémy interagují s webovou službou předepsaným způsobem pomocí SOAP zpráv, které jsou zpravidla zprostředkovány prostřednictvím HTTP s XML serializací ve spojení s dalšími souvisejícími webovými standardy[2]. Webové služby byly definovány dalšími způsoby, ale pro pochopení lze jednoduše formulovat webovou službu tak, že je to webová aplikace kladoucí si za cíl výměnu, zpracování a poskytování dat.

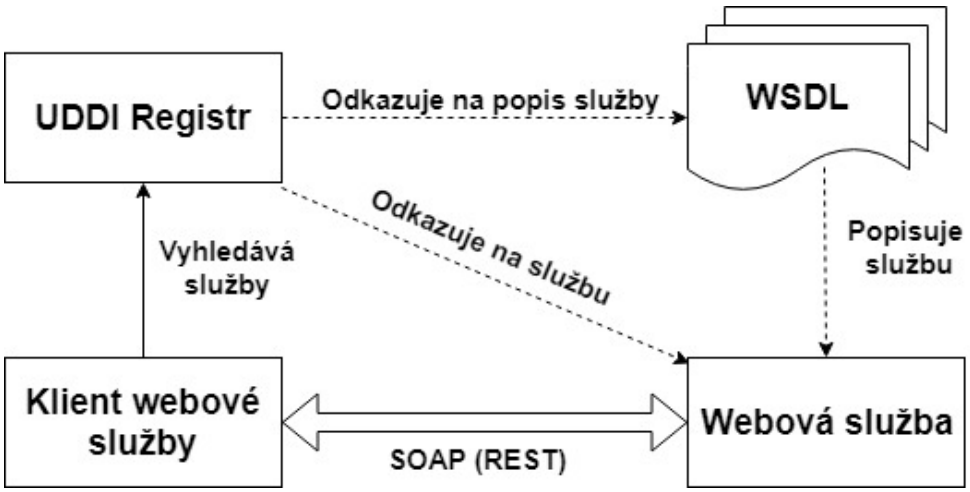
S pojmem webová služba taktéž úzce souvisí architektura. Architektura webových služeb udává abstrakci službám, jelikož většina služeb obsahuje stejné nebo podobné stavební prvky. Architektura si neklade za cíl specifikovat, jak má být služba implementována a jak mají být webové služby navzájem kombinovány. Míří na základní (minimální) charakteristiky pro veškeré služby a na jejich vlastnosti, které mohou, ale nemusí některé vyžadovat.

V dnešní době nechápeme webové služby obecně, jak zní definice W3C, ale díky zvýšeným potřebám trhu vznikají služby se specifickým zaměřením pro:

- vyhledávání
- počasí
- kurzovní lístky
- burzy
- bankovnictví
- ubytování
- ...

2.2 Základní funkční jednotky webových služeb využívající SOAP i REST

Webové služby jsou úzce spjaty s technologiemi SOAP a REST (popsány ve zvláštní kapitole). Následující schéma naznačuje strukturu webových služeb za použití SOAP nebo REST, kdy REST je novějším přístupem.



Obrázek 1: Architektura webových služeb s technologií SOAP

Na obrázku lze vidět celkem čtyři prvky: **UDDI registr**, **WSDL**, **Klient webové služby** a **Webová služba**. Mezi prvky probíhají různé interakce, kde má každý z nich pevně danou funkci.

2.2.1 UDDI registr

UDDI je schváleným standardem skupiny OASIS a klíčovými členy, zabývající se webovými službami. Definuje standardní metodu publikování a objevování síťově zaměřenými softwarovými komponentami založenými na servisně orientované architektuře (SOA).

Tato specifikace tedy definuje skupinu webových služeb a programových rozhraní pro publikování, získávání a správu informací o službách (v reálném SOA stylu je UDDI registr sám složen z webových služeb). UDDI je založen nad několika dalšími technologickými standardy, jako jsou HTTP, XML, XSD, SOAP a WSDL [3].

2.2.2 WSDL

Dle W3C WSDL (verze 2.0) poskytuje model a XML formát pro popis webových služeb (v práci dále jen SD z anglického “Service description“). WSDL umožňuje separovat popis abstraktní funkcionality nabízené službou od konkrétních detailů SD, jako jsou “jak“ a “kde“ je tato služba nabízena [4].

Zjednodušeně lze popsat WSDL jako jazyk, který abstraktně popisuje službu, stejně tak aplikační rámec (framework) pro detailnější popis této služby.

2.2.2.1 Příklad WSDL

Z důvodu rozsáhlosti XML formátu je příklad zkrácen pro ilustrativní účely.

```
<definitions name = "HelloService" targetNamespace = "http://.../HelloService.
    wsdl">
   <message name = "SayHelloRequest">
      <part name = "firstName" type = "xsd:string"/>
   </message>
   ...
   <portType name = "Hello_PortType">
      <operation name = "sayHello">
         <input message = "tns:SayHelloRequest"/>
         <output message = "tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name = "Hello_Binding" type = "tns:Hello_PortType">
      <soap:binding style = "rpc" transport = "http://schemas.xmlsoap.org/soap/
          http"/>
      <operation name = "sayHello">
         <soap:operation soapAction = "sayHello"/>
         <input><soap:body ... /></input>
         <output><soap:body ... /></output>
      </operation>
   </binding>

   <service name = "Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding = "tns:Hello_Binding" name = "Hello_Port">
         <soap:address location = "http://www.examples.com/SayHello/" />
      </port>
   </service>
</definitions>
```

Výpis 1: Ukázka WSDL služby

2.2.3 Klient webové služby

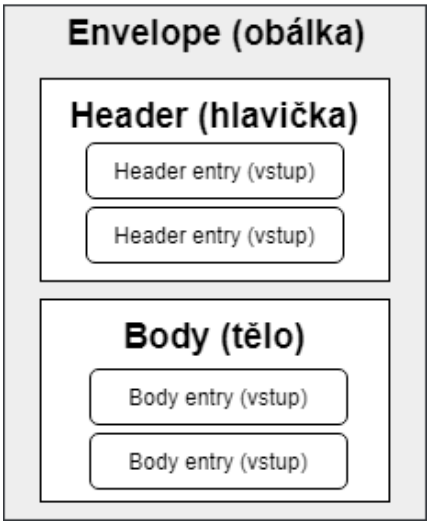
Klientem webové služby může být jakýkoliv software, který umí komunikovat po síti. Typické chování klienta je takové, že na základě určitých parametrů vyhledává služby skrze UDDI, a tím získá SD (tento krok může být vynechán, pakliže víme SD předem). Z SD je patrné, jak může klient komunikovat s webovou službou, tedy je známo, jakou má mít zpráva strukturu a kam ji zaslat.

2.2.4 Webová služba

Při vývoji webové služby je nutné definovat správně její popis. Dále je žádoucí vystavit službu okolnímu světu, neboli zaregistrovat službu do UDDI registru, který je pak schopen tuto službu nabízet. Toto samozřejmě platí pro globálně dostupné webové služby. Webová služba komunikuje s klientem prostřednictvím sítě na základě SOAP nebo REST technologie.

2.3 SOAP

SOAP (verze 1.2) poskytuje definici informací založených na formátu XML, které mohou být využity k výměně strukturovaně napsaných informací mezi dvěma body v decentralizovaném a distribuovaném prostředí. SOAP je ve svém základě bezstavové jednosměrné paradigma pro výměnu dat. Aplikace však mohou vytvářet více komplexních vzorů interakcí (např. požadavek/odpověď, požadavek/více odpovědí apod.). Protokol nespecifikuje sémantiku aplikačně specifických dat, jako jsou směrování přenosu dat, spolehlivost datového přenosu a firewallu. SOAP poskytuje rámec, podle kterého se mohou informace specifické pro danou aplikaci přenášet rozšířitelným způsobem [5].



Obrázek 2: SOAP model

SOAP lze tedy jednoduše popsat jako mechanismus pro výměnu dat ve formě XML přes síť. Tím SOAP získává benefit v podobě standardizovaného propojení mezi různými službami. Ve většině použití využívá protokolu HTTP (HTTP je prakticky základním kamenem dnešní internetové infrastruktury), což mu pomáhá v prostupování firewallem. Velkou nevýhodou je rozsáhlost zápisu, ale taky pomalé zpracovávání jednotlivými systémy, které musí provádět parsování a validaci.

SOAP se skládá ze tří částí: **obálky**, **hlavičky** a **těla**. Veškeré části lze spatřit ve výše vyobrazeném modelu. Obálka (envelope) je povinným elementem SOAP zprávy. Definuje, kde začíná a kde končí zpráva. Hlavička (header) je nepovinná a obsahuje atributy potřebné při zpracovávání zprávy. Tělo (body) obsahuje odeslanou zprávu ve formátu XML a je povinným článkem obálky.

2.3.1 Příklady požadavku a odpovědi na klientský požadavek

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetails xmlns="http://warehouse.example.com/ws">
       <productID>827635</productID>
     </getProductDetails>
   </soap:Body>
</soap:Envelope>
```

Výpis 2: Ukázka SOAP požadavku na službu

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
       <getProductDetailsResult>
         <productName>Bluetooth klavesnice</productName>
         <productID>827635</productID>
         <description>Podsvicena bluetooth klavesnice</description>
         <price>889,90</price>
         <inStock>ano</inStock>
       </getProductDetailsResult>
     </getProductDetailsResponse>
   </soap:Body>
 </soap:Envelope>
```

Výpis 3: Ukázka SOAP odpovědi na požadavek

2.4 REST

Pojem REST byl poprvé představen Royem Fieldingem v jeho dizertační práci[6], v které vysvětluje základní principy, kde definuje REST jako architektonický styl pro distribuované hypermediální systémy. Poskytuje množinu architektonických omezení takových, že při jejich použití zvyšujeme škálovatelnost interakcí mezi komponentami, zobecňujeme rozhraní, zlepšujeme nezávislost nasazení komponent, snižujeme latenci, vynucujeme bezpečnost a zapouzdřujeme starší systémy[6].

REST nebo taktéž RESTful (RESTful se stává po splnění veškerých omezení) webové služby je způsob komunikace mezi dvěma stroji skrze internet. Služby, které jsou kompatibilní s touto technologií umožňují přistupovat ke zdrojům pomocí uniformních a předdefinovaných bezstavových operací.

Samotné srovnání SOAP a REST není možné, jelikož SOAP je standardizovaný protokol a REST architektonický styl. Můžeme ale uvést hlavní body, v čem se tyto technologie liší, pakliže si uvědomujeme kontext obou technologií.

- REST není závislý na žádném protokolu, i když se využívá nejčastěji s HTTP. Pro provoz REST služby je nutné pouze standardizované URI schéma.
- REST je standardizovaný na základě technologií, které jsou využity v dané službě. Jestliže služba využívá HTTP, tak pro ni platí standardizace z HTTP, například bezpečnost a autentizace.
- Služba nemá architekturu REST bez hypermédia a HATEOAS přístupu.
- REST musí být postaven nad zdroji, nikoliv dokumentací [7].
- SOAP je pevně svázán se serverem.
- SOAP potřebuje znát vše, co bude potřebovat. REST poskytuje možnost kódu na požádání, kde klasickým příkladem je JavaScript poskytnutý serverem pro interakci na klientově straně.

2.4.1 Hypermédium

Hypermédium je rozšířením výrazu hypertext. Definuje se jako nelineární médium informací, které zahrnují grafiku, zvuk, video, prostý text a hypertextové odkazy.

2.4.2 HATEOAS

HATEOAS (z anglického “Hypermedia as the Engine of Application State“)je omezení aplikací na principu REST architektury, které REST rozlišuje od ostatních síťových architektur. Klient interaguje s webovou aplikací, která je dynamicky poskytována skrze hypermédium. Klient tak nepotřebuje znát, jak provést integraci s aplikací nad rámec chápání práce s hypermédiem.

2.4.3 Architektonické omezení

Následující část popisuje jednotlivé architektonické omezení pro REST služby. Pakliže služba tyto omezení dodržuje, nazývá se RESTful službou.

2.4.3.1 Klient-server architektura Síťová architektura klient-server odděluje klienta a server, kteří spolu mohou komunikovat prostřednictvím počítačové sítě. Toto oddělení je ve většině případů fyzické, ale může nastat situace, kdy komunikace probíhá na jednom stroji. Hlavní funkcí klienta je požadovat zdroje nebo služby ze serveru. Server tedy sdílí prostředky klientovi.

2.4.3.2 Bezstavovost Dalším omezením je bezstavovost. Toto omezení říká, že komunikace ve své podstatě musí být bezstavová. Každý požadavek klienta musí obsahovat kompletní informace k pochopení požadavku. Server neukládá žádná data o požadavcích klienta, vše obhospodařuje klient. Toto je taktéž jedna z největších nevýhod bezstavovosti. Každý požadavek musí být vykonán znovu opakovaně, takže nastávají redundantní volání na server, což zahlcuje síť. Částečné, ale nikoliv správné (potenciál špatné implementace), řešení této situace je takové, že si sám klient uchovává některá data.

Díky tomuto omezení je dosaženo třech významných vlastností: viditelnost, spolehlivost a škálovatelnost. Viditelnost je odvozena od kompletního zasílání informací klientem, není tedy potřeba zjišťovat pravou podstatu žádosti. Spolehlivost zajišťuje snadné zotavování z částečných poruch. Tím, že není potřeba ukládat stav žádosti, je vylepšena škálovatelnost. Server tedy může rychle uvolňovat paměť.

2.4.3.3 Ukládání do mezipaměti - cache Kvůli zefektivnění síťové komunikace zavádíme omezení pro mezipaměť. Požadavky jsou ukládány do mezipaměti implicitně nebo explicitně a jsou označeny “pro ukládání do mezipaměti“ nebo “bez ukládání do mezipaměti“. Klient tak může obdržet data z mezipaměti pro urychlení komunikace.

Výhodou je potenciální snížení interakce, a tím zvýšení efektivity, škálovatelnosti a výkonu snížením latence sítě. Problémem je však snížení spolehlivosti z důvodů možné práce s neaktuálními daty.

2.4.3.4 Vrstvený systém Umožňuje hierarchickou strukturu architektury. Klient není schopen říci, zda je připojen přímo ke koncovému serveru nebo ke zprostředkovateli cesty ke zdroji. Interaguje pouze s nejbližší vrstvou.

Jednotlivé vrstvy mohou mít různou funkcionalitu, například zaobalení starších služeb, a tím zajištění ochrany nové službě před starými klienty. Může taktéž zlepšit škálovatelnost zlepšeným vyvažováním zátěže služby mezi více sítěmi a procesory. Majoritní nevýhoda spočívá v tom, že zvyšuje režii a latenci pro zpracování dat.

2.4.3.5 Kód na požádání REST umožňuje rozšiřování funkcionality stažením a spuštěním kódu ve formě skriptu nebo appletů. Tím snižuje nutnost mít dříve implementovanou funkčnost a zajišťuje rozšiřitelnost, ale také snižuje viditelnost. Toto omezení je volitelné, není nutné jej dodržet, avšak mnoho dnešních aplikačních rámců tuto funkcionalitu podporuje.

2.4.3.6 Uniformní rozhraní Klíčový bod REST architektonického stylu je uniformita rozhraní. Zobecněním přístupového bodu k aplikaci je zjednodušena architektura systému a vylepšena viditelnost interakcí možných se systémem. Nezávislost vývoje podporuje to, že implementace nejsou svázány se službami, které poskytují.

Obecnost může způsobovat snížení efektivity. Pro zachování správné funkce rozhraní by měly být dodrženy čtyři omezení: identifikace zdrojů (pomocí URI), manipulace se zdroji prostřednictvím reprezentací (JSON, XML apod.), samo popisující zprávy (dostatek informací k pochopení) a HATEOS (vysvětlen v samostatné části).

2.4.4 WADL

WADL, stejně jako WSDL, je strojově čitelný jazyk ve formátu XML sloužící k deskripci webových aplikací, které pracují prostřednictvím protokolu HTTP (poskytují svá interní data přes HTTP). WADL si klade za cíl modelovat prostředky poskytované aplikacemi spolu se vztahy mezi jednotlivými aplikacemi a znovupoužitelností webové aplikace (služby). Stejně jako WSDL není WADL závislý na jakémkoliv jazyce. Dále je nutné zmínit, že WSDL 2.0 lze taktéž využít pro HTTP založené aplikace.

2.4.4.1 Formát W3C popisuje čtyři hlavní aspekty strojově čitelného formátu:

- **Zdroje**, analogicky k mapě stránek, která ukazuje nabízené zdroje.
- Popisující **vazby mezi zdroji**, a to jak referenční, tak příčinné souvislosti.
- HTTP **metody aplikovatelné na zdroje**, očekávané vstupy, výstupy a podporovaný formát.
- **Formáty reprezentace zdrojů**. Podporované MIME typy a používáané datové schémata.

Zkráceně lze popsat WADL tak, že poskytuje strojově zpracovatelný jazyka pro webové aplikace pracující s HTTP.

2.4.4.2 Nadbytečnost nebo nezbytnost pro REST

Jedním z principů REST architektury je ten, že využívá již existující webovou infrastrukturu. Díky tomu může (například pomocí HTTP protokolu) komunikovat po síti standardizovaným způsobem. RESTful služba by měla být bezstavová, klient-server apod. Z toho vyplývá, že je služba jasně specifikována a nepotřebuje další popis. Zjednodušeně se dá říci, že RESTful služba, která je založena na HTTP protokolu, snadno identifikuje sama sebe již díky určeným pravidlům (viz výše). Jak ale již z nadpisu vyplývá, tak některé situace si žádají, aby služba poskytovala svůj popis (SD). Setkáváme se s řadou názorů, které vyvracejí nutnost použití SD pro RESTful služby z důvodů, že RESTful služba je sebepopisující. Můžeme se setkat taktéž s oponenty předchozího názoru, kteří uvádí jako protiargument enormní složitost dnešních podnikových aplikací. Zde je příklad, kdy použít, a kdy nepoužít SD pro RESTful služby:

- **SD nepotřebujete**, pakliže vyvíjíte novou webovou aplikaci úplně od začátku bez závislostí.
- **SD nepotřebujete**, pakliže integrujete váš systém s jiným systémem za situace, kdy jste v kontaktu s autorem, který vyvíjel tuto aplikaci.
- **SD potřebujete**, pakliže integrujete komplexní systém s více jinými systémy (např. vládní systémy, systémy mnoha různých společností apod.), kdy s velkou pravděpodobností nebudete v kontaktu s autory těchto systémů.
- **SD potřebujete**, pakliže integrujete systém s nějakým starým, již nepodporovaným systémem. Zde potřebujete vědět, jak komunikovat s tímto systémem. Řešením je mít SD tohoto systému.

2.4.4.3 Příklad WADL

Z důvodu rozsáhlosti XML formátu je příklad zkrácen pro ilustrativní účely.

```
<application xmlns="http://wadl.dev.java.net/2018/05">
  <resources base="http://example.com/api">
    <resource path="books">
      <method name="GET"/>
      <resource path="{bookId}">
        <param required="true" style="t" name="bookId"/>
        <method name="GET"/>
        <resource path="reviews">
          <method name="GET">
            <request>
              <param name="page" required="false" default="1" style="q"/>
              <param name="size" required="false" default="20" style="q"/>
            </request>
          </method>
        </resource>
      </resource>
    </resource>
  </resources>
</application>
```

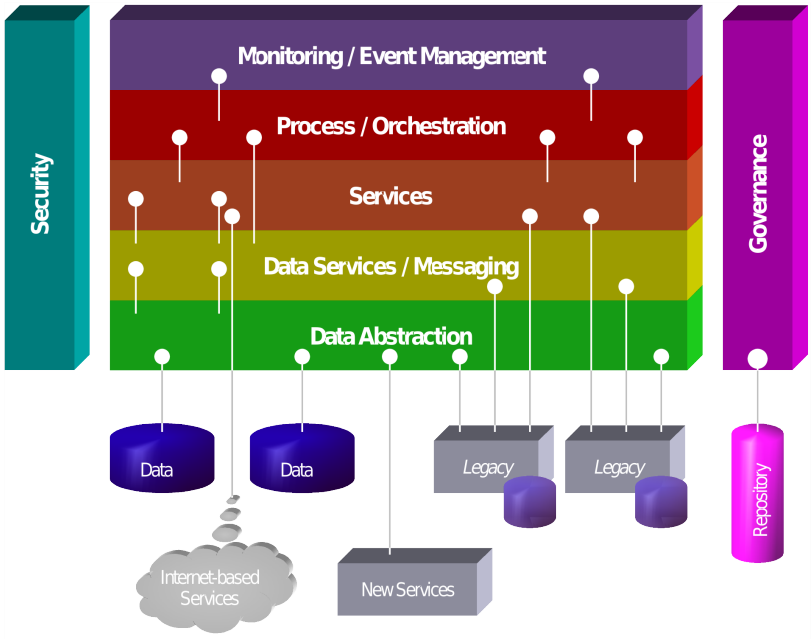
Výpis 4: Ukázka WADL

2.4.4.4 Shrnutí

WADL byl v počátku alternativou WSDL (určený původně pro SOAP) pro REST. Jeho velkým nedostatkem je nestandardizovaný rámec. Můžeme se tedy setkat s různými verzemi, které nejsou navzájem kompatibilní. Naopak tento nedostatek je u RESTful služeb vynahrazen právě sebepopisující schopností služby. Toto však nenabízí WADL, nýbrž REST architektura, takže z tohoto důvodu se doporučuje v dnešní době využívat WDSL 2.0. WSDL 2.0 umožňuje popisovat i REST služby, takže se stal obecnějším, než WADL. Využití jakékoliv obecné metody k pro SD v REST je nutné zvážit, protože REST je koncipován jako sebepopisný, ale existují i situace, kdy je nutné SD zavést.

2.5 Servisně orientovaná architektura

V dnešní době cloudových služeb se vývoj softwaru odebírá směrem ke zvýšení dynamiky, který zahrnuje zvýšení efektivnosti, rychlosti implementace, nasazení apod. Nově vytvářené aplikace bývají ve většině případů postaveny na principu softwaru jako služby (SaaS) za pomoci servisně orientované architektury (SOA). Tato architektonická platforma je málo provázaná část softwarového systému, která zapouzdřuje svou logiku tak, že je snadno kombinovatelná a slučitelná s dalšími řešeními. Ačkoliv vytvářet aplikace, které umožňují vzdálený přístup k prostředkům a funkcionalitě jiných služeb není nikterak nová myšlenka, tak principy, jako malá provázanost, značí relativně nový přístup k vytváření kompozitních řešení.



Obrázek 3: SOA meta-model [8]

Definice dle opengroup [9]: Servisně orientovaná architektura je architektonický styl, který pomáhá k orientaci na služby. Orientace na služby je způsob přemýšlení nad službami, jejich vývojem a jejich výstupem. Služba:

- je logickou reprezentací opakovatelné byznys aktivity se specifickým výstupem (výše zmíněné typy služeb)
- je samostatná
- může být složena z dalších služeb
- je tzv. “black box“ pro spotřebitele služby

2.6 Popis služby - Service description (SD)

Popis webových služeb hraje důležitou roli v přístupnosti k dané službě (viz část 2.4.4). Důvodem vzniku SD bylo výhradně poskytnutí informací o službách ve formě strojově čitelného jazyka, což umožňuje následnou automatizaci, snazší vyhledávání a klasifikaci webových služeb. Veškeré tyto aspekty jsou dále využity v obecné kompozici služeb.

Obrázek 4: Využití SD

2.6.1 Strojově čitelná nebo člověkem čitelná struktura SD

Obě struktury z nadpisu jsou běžnou součástí SD. Člověkem čitelná forma značí, že je služba popsána pochopitelným pro člověka bez nutnosti znát nějaké technologie, například národní jazyk. Tuto formu, často poskytovanou poskytovatelem nebo vývojářem služby, začleňujeme jako dokumentaci pro orientaci potencionálních žadatelů o službu. Strojově čitelná forma SD je WSDL (WADL apod.). Tato reprezentace umožňuje automatické čtení, zpracování a využití kvalifikovanými nástroji (wsimport).

Vzájemný vztah mezi oběma přístupy (stroj versus člověk) připomíná vztah Webu a Sémantického Webu. Web vytvořený s HTTP a URI je informační prostor pro lidskou spotřebu. Jeho statický obsah (dokumenty) je napsán v přirozených jazycích pochopitelných pro lidi. Sémantický Web je rozšíření současného webu, ve kterém mají informace přiděleny dobře definovaný význam pro lepší kooperaci počítačů a lidí [10]. Je zřejmé, že obsah Sématického Webu není určen pro člověka, ale pro stroje, které pracují na sofistikovaných úkolech zadaných člověkem.

2.6.2 Výzkumné metody nebo průmyslové standardy pro SD

V současné době existuje mnoho přístupů pro popisování služeb. Z pohledu využitelnosti je dobré přistupovat k rozdělení dvěma způsoby[11], kde každý přístup má své benefity. Prvním jsou **výzkumné metody**. Tyto metody se orientují na různé aspekty, například bezpečnost, QoS atd. Důležitým faktorem je to, že tento přístup je vhodný pro nestandardizované softwary. Nestandardizovanými softwary (službami) rozumíme taková softwarové díla, které nejsou nikterak formálně (např. právně) závislá na používaných zdrojích, nebo pro ně neplatí žádná norma. Takovouto službu si můžeme představit jako aplikaci, která poskytuje informace o našich zdrojích. Přístup **průmyslových standardů** vyhovuje zejména tam, kde je potřeba jasné specifikace služby. Tento přístup je detailnější, a tím vyžaduje vyšší specializaci na zdroje a formalizaci. Jednoduchým příkladem může být služba, která zajišťuje funkcionalitu, která může ohrozit člověka na životě (strojnictví, automotive apod.) [11][12].

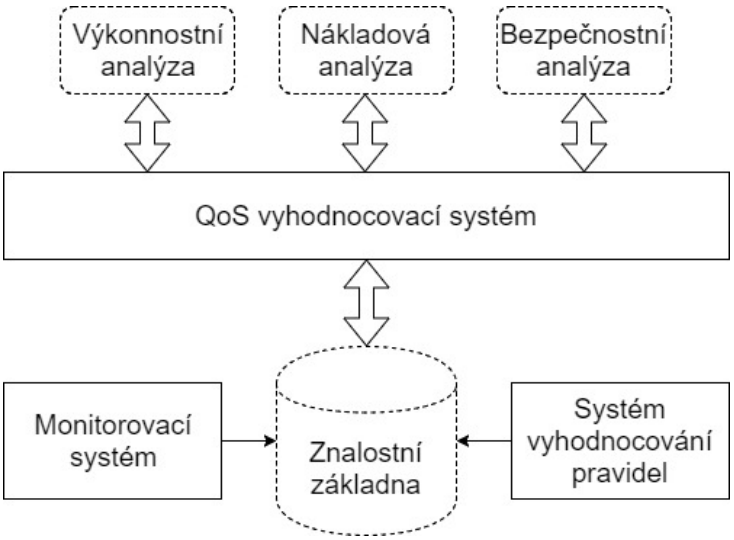
2.6.3 Shrnutí

Lze tedy zkráceně říci, že výzkumné metody se zaměřují jen na některé, pro ně klíčové, parametry SD, a tím jsou snazší pro vývoj, kdežto průmyslové standardy musejí být unifikované, obecné a zaměřují se na co nejmenší chybovost. Tento způsob je velmi náročný a vyžaduje širokou analýzu.

2.7 Kvalita služby - QoS

Vzhledem k tomu, že se webové služby staly nejpoužívanějším prostředkem pro vytváření softwarových aplikací, stále více lidí vyhledává praktické aspekty pro využitelnost v byznyse. S enormním využíváním služeb se jejich počet neustále navyšuje, a kvůli toho je velmi obtížné se v nich zorientovat. Jedním pomocným atributem služby je její kvalita. Dnes je kvalita asi nejzmiňovanější věc v IT světě. Kvalita služeb (QoS) zapouzdřuje velkou množinu atributů, které určují službě tzv. skóre. Toto skóre je možné definovat různými potřebami, takže není možné ho zobecnit přes všechny parametry, protože je zde možnost definování si vlastního parametru. Díky QoS se zjednodušil mechanismus pro vyhledávání vyhovující služby.

Zjednodušeně lze poznamenat, že **QoS umožňuje výběr nejlépe vyhovující služby** z potenciální množiny velmi podobných služeb.



Obrázek 5: Architektura vyhodnocování kvality webové služby [13]

2.7.0.1 Vyhodnocování kvality webové služby Na obrázku architektury vyhodnocování kvality webové služby lze vidět kategorizaci na tři hlavní oblasti kvality: **výkonnostní analýza**, **nákladová (cost) a bezpečnostní analýza**. Dále je zde **monitorovací systém**, který obecně vyhodnocuje a ukládá parametry do znalostní základny. **Znalostní základna** ukládá vyhodnocené parametry služeb. Na základě těchto uložený informací **systém vyhodnocování pravidel** vybere nejvhodnější službu. Tento systém vyhodnocuje námi vybrané důležité parametry.

2.7.1 Soubor několika QoS parametrů služeb

Parametrů služeb existuje nespočet. V následující tabulce jsou shrnuty nejpoužívanější z citovaných zdrojů. Faktory jsou obecnější parametry, které zapouzdřují subfaktory. V praxi jsou tyto faktory i subfaktory provázány [14].

Faktor	**Související subfaktor**
Udržitelnost	Opravitelnost
	Rozšiřitelnost
	Testovatelnost
Přenositelnost	Nezávislost hardware
	Nezávislost software
	Instalovatelnost
	Znovupoužitelnost
	Rozpoznatelnost
Efektivnost	Hospodaření s časem
	Hospodaření se zdroji
Spolehlivost	Robustnost
	Tolerance chyb
	Dostupnost
Funkcionalita	Úplnost
	Správnost
	Kompatibilita
	Interoperabilita
	Bezpečnost
	Výkonnost

Tabulka 1: Nejběžnější faktory QoS a související subfaktory [13]

2.8 Bezpečnost

Téma bezpečnosti je s rostoucím počtem útoků neustále více skloňováno v IT světě. Jelikož nejnovější architektury jsou založeny na technologii webových služeb, platí pro webové služby taktéž zásady bezpečnosti. Bezpečnost webových služeb, stejně jako SD pro REST, není nikterak standardizována. Vývojáři dělají různá ad-hoc řešení, a tím se velmi často vyskytují problémy s interoperabilitou. Proto je nutné se o tomto tématu zmínit, jelikož zabezpečení je neméně důležitým prvkem, jakým byl SD. Pro vývojáře webových služeb je bezpečnost důležitým kontextem vývoje. Komunikace může probíhat skrze různé prostředníky (další služby), což není obvyklé u síťových služeb. Komunikace využívá jak PPP, tak E2E komunikace (může i zároveň).

Vrstvy, na které je nutné brát zřetel z pohledu bezpečnosti [15]:

- **Transportní** vrstvu ošetřují firewally, virtuální privátní sítě, jednoduché ověřování, nepopiratelnost a šifrování.
- Vrstvu **zpráv**, zabezpečují ověřovací tokeny pro identifikaci žadatele a autorizaci žadatele k řízení přístupu ke službě.
- **Datová** vrstva zajišťuje šifrování dat a digitální podpis pro ošetření manipulace s daty.
- Vrstva **prostředí** poskytuje správu, protokolování a audit pro vyřešení vyskytnutých problémů. Vytváří také důvěryhodné vztahy a komunikační vzory.

Pro zabezpečení webových služeb je zapotřebí řady bezpečnostních mechanismů (často založených na XML), které řeší problémy související s autentifikací, řízením přístupu založeného na rolích, distribuováním bezpečnostní politiky, vrstvou zabezpečení zpráv [2].

2.8.1 Bezpečnostní politiky

Bezpečnostní politika, je/jsou strojově zpracovatelné soubory, popisující omezení na zdrojích. Dělí se na dva typy: **politiku oprávnění** a **povinných zásad**. První z nich se týká akcí a přístupů, do kterých mohou subjekty zasahovat. Druhý z nich se týká akcí a stavů, které jsou subjekty nuceny provádět. Soubory, obsahující bezpečnostní politiku, pravděpodobně zahrnují oba tyto typy, jelikož jsou úzce spjaty (nemůžeme být nuceni provádět nějakou akci, na kterou nemáme práva) [2].

2.8.2 Hrozby pro vrstvu zabezpečující zprávy

Z možnosti E2E komunikace je možné vystavit webovou službu řadě útoků během zasílání zpráv. Je nezbytné tyto hrozby zmínit a upozornit na možné zneužití služeb [2]:

- **Změna zprávy** je technika, kdy útočník změní (upraví, smaže nebo přidá nová) data, a tím naruší integritu zprávy.
- **Narušení důvěrnosti** může zpřístupnit důvěrná data nežádoucí osobě.
- **MITM**, kdy útočník má kompletní přístup ke komunikaci mezi stranami, které o jeho zapojení neví.
- **Spoofing**, kdy útočník přejímá identitu důvěryhodného subjektu za účelem sabotáže zabezpečení cílové entity.
- **DoS** zabraňuje oprávněným uživatelům služby ji využívat (např. vytížením této služby).
- **Útok přehráním**, data mohou být při přenosu zachycena a později je mohou být zopakována nebo pozměněna při průchodu síťovými prvky.

2.8.3 Klíčové bezpečnostní požadavky

V předchozím bodě byly zmíněny hrozby pro webové služby. Aby bylo riziko hrozeb co nejnižší, je zapotřebí klást určité požadavky na webové služby.

- **Autentifikační mechanismy**
- **Autorizace**
- **Integrita dat a důvěrnost údajů**
- **Integrita transakcí a komunikace**
- **Nepopiratelnost původu**
- **Celistvost a důvěrnost zpráv**
- **Audity**
- **Distribuované vymáhání bezpečnostních politik**

3 Kompozice webových služeb

S pokračujícím pokrokem v oblasti webových služeb vznikla potřeba interoperability mezi jak již existujícími, tak mezi nově implementovanými službami. Skládání, neboli jiným názvem kompozice je technika k zajištění interoperability mezi službami. Služby mezi sebou interagují za nějakým cílem. K pochopení samotné kompozice je potřeba znát alespoň stručný přehled a kontext technologií webových služeb, což zajišťuje předchozí část této práce.

V kontextu paradigmatu webových služeb je kompozice mechanismus kombinující více služeb k vytvoření komplexnějších funkcionalit. Zvyšuje tím potenciál jednotlivých služeb [16]. Kompozice může pracovat na dvou principech: kompozici služeb, nebo kompozici služeb, které jsou již v nějaké kompozici [17].

Kompozici lze chápat jako techniku, která prostřednictvím propojení služeb vede k dosažení požadovaného cíle. Tímto cílem si můžeme představit například webovou službu pro objednání zásilky, která se může skládat z dalších různých podpůrných služeb zajištujících koncovou správnost výsledku.

3.1 Statická a dynamická kompozice

Na základě doby, kdy jsou webové služby skládány, dělíme kompozici na statickou a dynamickou. Při **statické** kompozici se agregace služeb uskutečňuje v době návrhu. Požadované součásti služby jsou vybrány, svázány a nasazeny. Statická kompozice funguje dobře, pakliže jsou všichni obchodní partneři relativně fixně zahrnutí do procesu a funkční požadavky se málo mění. Statická kompozice není flexibilní a adaptabilní v případě častých změn požadavků, které nelze předvídat v době návrhu. Jakmile se služba stane nedostupnou, nebo existuje lepší alternativní služba k dané službě, statická kompozice nemůže poskytnout dostatečnou podporu při běhu. Naproti tomu **dynamická** kompozice umožňuje určování a nahrazování součástí služby během provozu. Dynamická kompozice požaduje, aby systém umožňoval automatickou detekci, výběr a přiřazení komponent služeb [18].

Dynamická kompozice je tedy flexibilní a adaptabilní. Dokáže se velmi rychle přizpůsobit změně požadavků s minimálními změnami. Prostředí webových služeb je dynamické, a proto je tedy lepší volbou dynamická kompozice, avšak dynamická kompozice je velmi náročným úkolem, kde je třeba zvážit velké množství parametrů (časové limity, chybové stavy, podpora transakcí, korektnost výsledků, využití zdrojů apod.).

3.2 Manuální, poloautomatická a automatická kompozice

Kompozitní služby velmi často zahrnují komplexní obchodní (byznys) procesy, které se mohou skládat z mnoha úkolů a interakcí mezi úkoly (řízení a tok dat, řešení závislostí transakcí, řešení chybových situací apod.). Implementace kompozitních služeb je časově i zdrojově náročná, proto v současnosti existují tři typy řešení kompozice služeb [18].

3.2.1 Manuální kompozice

Manuální kompozice je nejsnazším řešením kompozice služeb. Spočívá v tom, že člověk, který službu poskytuje, vytvoří abstraktní kompozitní proces za pomoci standardizovaných jazyků pro webové služby (BPEL). Následně naváže jednotlivé webové služby do tohoto procesu manuálně, což vyžaduje kompletní znalost dané domény. To je důvod, proč je toto řešení časově neefektivní. Dalším záporem manuální kompozice je potenciál vytvoření chyb, takže se nemůžeme spolehnout na výsledek procesu, nemáme tedy žádnou garanci správnosti.

3.2.2 Poloautomatická kompozice

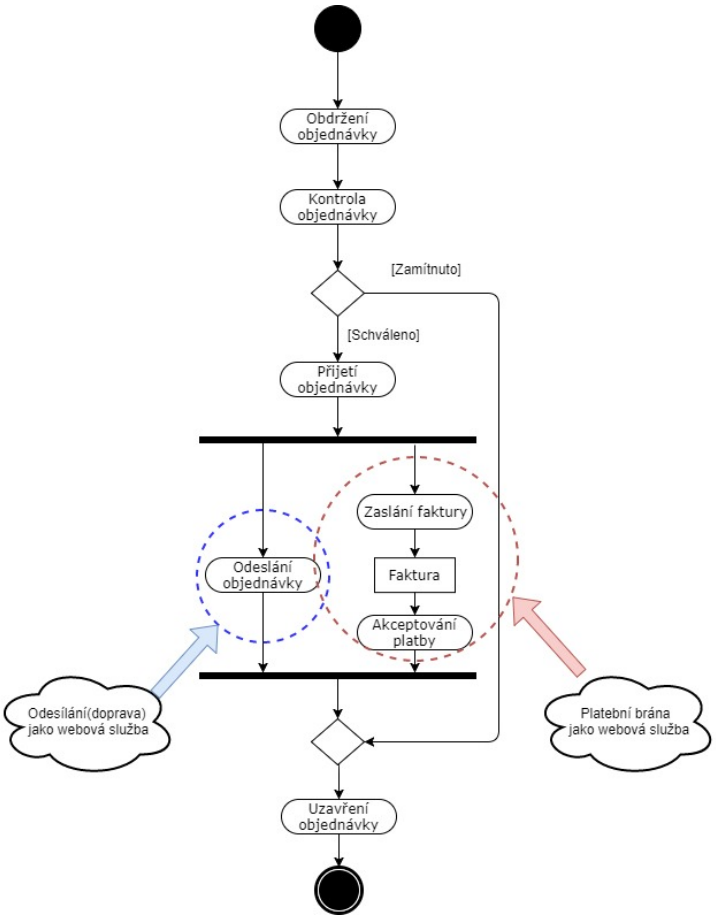
Poloautomatická kompozice spojuje prvky automatické a manuální kompozice. Tuto kompozici lze popsat tak, že člověk, který službu vytváří, nemusí být do samotné kompozice (popisu procesu) začleněn. Proces se může automaticky vytvořit na základě specifikací služby (například automaticky vygenerovat soubory v BPEL pro popis všech procesů). To umožňuje přiřazovat jednotlivé webové služby za běhu. Člověk do procesu vstupuje pouze s určitými parametry, podle kterých se řídí výběr služeb do procesu. Může definovat QoS parametry a omezení pro daný proces.

3.2.3 Automatická kompozice

Automatická kompozice je druh sémantické integrace systémových služeb. Takováto kompozice velmi často využívá technik Sémantického Webu. V dnešní době se do tohoto procesu začleňuje i umělá inteligence. Teorie spočívá v tom, že na základě dodané množiny služeb a specifikovaných požadavků, může být kompozitní služba vygenerována automaticky. Toto je velmi složitý proces s velkou řadou problémů. Největším problémem je to, že takovéto webové služby plně nevyužívají potenciál sémantiky (nedostatečný popis apod.), což ovlivňuje automatickou selekci správných služeb. Cílem této práce je průzkum automatické kompozice, a proto se ji budeme věnovat podrobněji v samostatné kapitole.

3.3 Příklad procesu objednání s náznakem webových služeb

Důvod k zavedení kompozice webových službeb může být ilustrován na procesu objednání zásilky. Proces nevyužívá webové služby, což značně omezuje využití implentovaných komponent. Pokud převedeme určité bloky na webové služby, tak následně můžeme tyto komponenty abstrahovat, čímž zajistíme jejich znovupoužitelnost a snadnou nahraditelnost. Díky zobecnění daných komponent můžeme, podle určitých pravidel, vybírat vhodné webové služby, což rozšiřuje možnosti celého procesu.



Obrázek 6: Potenciální využití webových služeb v částech procesu objednávky

Proces objednání zásilky má jasně daný průběh (v zjednodušeném příkladě), ale již zde lze najít určité prvky pro abstrakci, a tím i možnost znovupoužitelnosti nejenom v rámci jednoho procesu. Fakturace, jako webová služba, by mohla být schopna fakturovat jakékoliv zboží a proces odesílání může být realizován prostřednictvím webové služby dopravce. Celý proces objednání zboží může být webovou službou, která zapouzdřuje další webové služby (kompozice). Tato služba bude znovupoužitelná, a tím může být zautomatizována, jelikož v každém internetovém obchodě je nutné zboží fakturovat a doručovat.

4 Automatická kompozice webových služeb

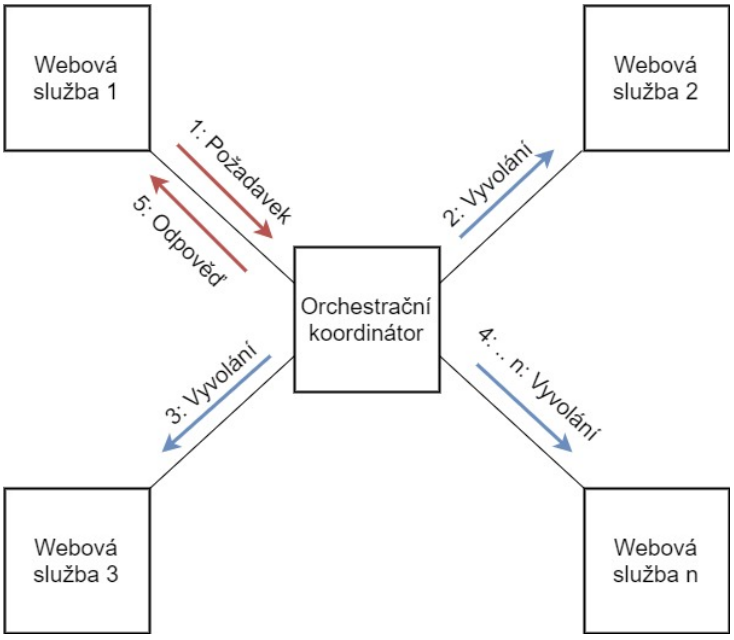
Kompozice úzce souvisí se dvěma pojmy, a to s **orchestrací** a **choreografií**. Každý z nich má jiný případ užití a konkrétněji budou obě techniky popsány dále. Pro úvod postačuje si představit orchestraci jako orchestr vedený centrálně dirigentem a choreografii jako sled decentralizovaných, na sebe navazujících aktivit.

4.1 Motivace

Motivací pro zavedení kompozitních služeb jsou v dnešní době tzv. cloudové služby. Podniky se snaží nabízet různou funkcionalitu zákazníkům tak, aby byla snadno použitelná a dostupná prostřednictvím počítačové sítě, pouze se změnou konfigurací dané služby (není vždy nutné mít různé konfigurace). Tím, že je služba zaměřená na jeden konkrétní cíl a má jasně daný popis, se dá snadno zahrnout do kompozice.

4.2 Orchestrace

Při orchestraci přebírá centrální proces (což může být další webová služba) kontrolu nad zainteresovanými webovými službami a koordinuje provádění různých operací webových služeb, které jsou součástí dané operace. Zúčastněné webové služby "nevědí" (a nemusí vědět), že jsou zapojeny do procesu kompozice, a že se účastní většího, výše postaveného podnikového procesu. Pouze centrální koordinátor orchestrace si je vědom tohoto cíle. Orchestrace je centralizovaná s explicitními definicemi operací a pořadím invokací webových služeb. Orchestrace se běžně využívá v privátních byznys procesech [19].



Obrázek 7: Schéma orchestrace

4.2.1 Příklad využití v praxi

Orchestraci lze popsat například procesem vybírání cesty do zahraničí. Každý uživatel postupuje podobným procesem. Vybírá destinaci, předpokládanou cenu, hotel, služby hotelu apod. Těchto parametrů mohou být desítky až stovky, v závislosti na požadované akci. Nejjednodušší způsob je zajít do cestovní kanceláře a nechat tento výběr na pracovníkovi kanceláře. S rostoucí digitalizací i zde můžeme vidět prvky, které lze převést do počítačové formy. Pracovníka cestovní kanceláře můžeme v kontextu kompozitních služeb považovat za centrálního koordinátora, a jednotlivé parametry za kompozitní webové služby. Pokud dodáme centrálnímu prvku předepsaný seznam požadovaných služeb, centrální prvek by měl být schopen komunikovat s dalšími službami pro poskytnutí informací potřebných pro klienta. Nemusíme si představit pouze služby cestovní kanceláře, ale můžeme se posunout o úroveň výše, kde si vybíráme leteckou společnost, dopravu na letiště a z letiště, aktivity během pobytu apod. Toto naznačuje, že služba pro výběr hotelu, se skládá z dalších služeb a sama je tedy kompozitní službou v procesu cestování do zahraničí. Každá služba má své určité parametry, které jsou kontrolovány na základě definovaného procesu.

4.2.2 Standardy orchestrace webových služeb

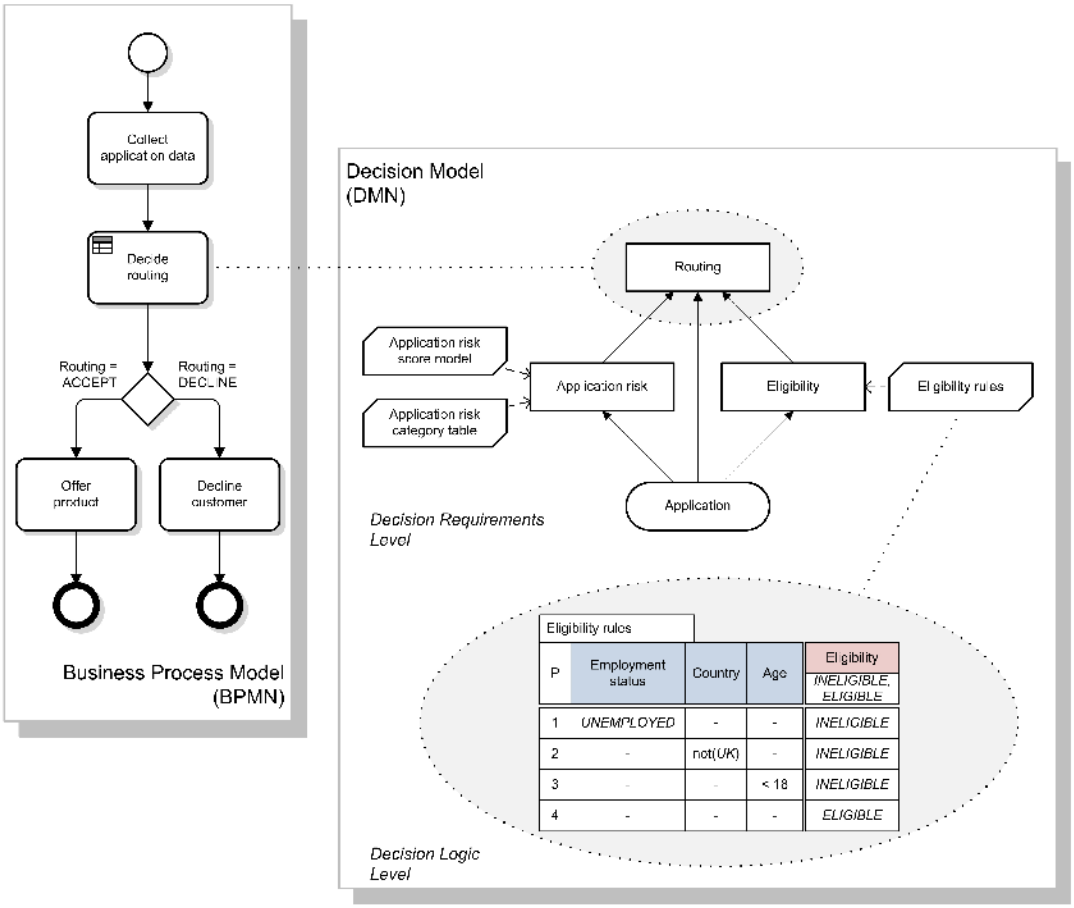
Jak již bylo popsáno výše, orchestrace je centralizované řízení procesu. Proces orchestrace zahrnuje řízení transakcí mezi jednotlivými službami, řešení chybových stavů a celkově popisuje komplexní proces. Při orchestraci využíváme dva standardy, které jsou úzce spjaty. BMPN definuje vizuální reprezentaci toku procesu. WS-BPEL (dále jen BPEL) je prováděcí jazyk, kdy pomocí kódu spouštíme webové služby.

4.2.2.1 BPMN Je soubor principů a pravidel sloužících ke grafické reprezentaci podnikových procesů prostřednictvím procesních diagramů [20]. Diagramy jsou souhrn aktivit, které prochází tzv. tokem, který je založen na různých okolnostech ovlivňujících daný proces. BPMN notace byla vytvořena skupinou Object Management Group s cílem poskytnout standardizovaný zápis procesu v přehledné a obecné formě. BPMN je komplexní a intuitivní. Právě proto se stal nejpoužívanějším standardem a v budoucnu pravděpodobně nahradí i prováděcí jazyky jako BPEL.

4.2.2.2 BPEL Je jazyk používaný ke kompozici, orchestraci a koordinaci webových služeb. Poskytuje bohatý slovník pro vyjádření chování podnikových procesů [19]. BPEL definuje model a gramatiku popisující chování podnikového procesu, založeného na interakci mezi procesem a jeho partnery. Interakce s každým partnerem probíhá prostřednictvím rozhraní webové služby a struktura tohoto vztahu je zapouzdřena v partnerLink. BPEL také zavádí systematické mechanismy pro řešení obchodních výjimek a poruch při zpracování. Definuje také to, jak mají být jednotlivé činnosti kompenzovány v případech, kdy dochází k výjimkám nebo při partnerském požadavku na zrušení činnosti [21].

4.2.2.3 Vztah BPMN a BPEL Vztah BPMN a BPEL se postupně měnil. V prvních verzích se oba jazyky doplňovaly, kde BPMN naznačoval grafickou strukturu procesu a BPEL tuto strukturu prováděl. Od verze 2.0 je v BPMN jazyk schopný vytvářet XML, které se následně provádí. Dnes se nejčastěji využívá BPMN konverze na BPEL, který je ideální k orchestraci služeb. BPEL od dalších verzí obsahoval grafický popis, avšak není standardizovaný jako BPMN, a proto existuje mnoho řešení s různými grafickými notacemi. To je důvod, proč se dnes využívá **BPMN s následnou konverzí na BPEL** (pokud je to potřebné).

4.2.2.4 DMN DMN je standardem, který doplňuje BPMN. Účel DMN je poskytovat konstrukce, které jsou potřebné k modelování rozhodování tak, aby se organizační rozhodování dalo snadno zobrazit v diagramech, které jsou přesně definovány obchodními analytiky a popřípadě automatizovány [22]. Na následujícím obrázku lze vypozorovat vztah DMN s BPMN. DMN se skládá ze dvou úrovní: **požadavků na rozhodnutí** a **logiky rozhodování**. Obě tyto úrovně mohou být použity samostatně nebo ve spojení pro modelování domény rozhodování, bez vztahu na podnikový proces.



Obrázek 8: Vztah BPMN a DMN [22]

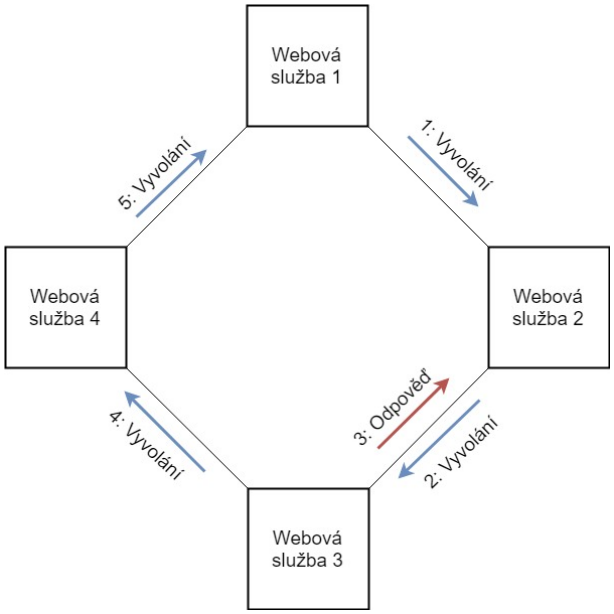
4.2.2.5 Výběr volně dostupných modelovacích nástrojů pro BPMN 2.0

Název produktu	**Aktuální verze**	**Platforma**
Activity	2017	multiplatformn
ARCWAY Cockpit	2017	Windows, Linux a MAC
AuraPortal	2017	Windows
BeePMN	2018	Cloud
Bizagi Modeler BPMN	2016	Windows
Bonita	2018	Windows, Linux a MAC
bpmn.io	2018	multiplatformní a cloud
BPMN Modeler for Confluence	2017	multiplatformní a cloud
Camunda	2017	multiplatformní
Cubetto	2017	Windows, Linux, Android a MAC
Eclipse BPMN2 Modeler	2017	multiplatformní
GenMyModel	2016	multiplatformní
HEFLO	2017	multiplatformní
Imixs Workflow	2018	multiplatformní
Innovator for Business Analysts	2018	Window
jBPM	2018	multiplatformní
LucidChart	2018	multiplatformní
Yaoqiang BPMN Editor	2017	Windows, Linux a MAC
yEd	2017	Windows, Unix/Linux a MAC
yEd Live	2017	multiplatformní

Tabulka 2: Výběr volně dostupných modelovacích produktů pro standard BPMN 2.0 aktivních od roku 2015 [23][24][25]

4.3 Choreografie

Choreografie se oproti orchestraci nespoléhá na centrálního koordinátora. Každá webová služba, která se podílí na choreografii přesně ví, kdy provést své operace, a s kým komunikovat. Choreografie je společné úsilí zaměřené na výměnu zpráv ve veřejných obchodních procesech. Všichni účastníci choreografie si musí být vědomi celého obchodního procesu, operací k provádění, zpráv ke komunikaci a načasování výměny zpráv.



Obrázek 9: Schéma choreografie [19]

Z obou schémat typů řízení je patrné, že orchestrace je více používána než choreografie. Orchestrace je více flexibilní paradigma, ale při některých situacích je použitelnější choreografie. Při orchestraci přesně víme, kdo je zodpovědný za celý proces. Může začleňovat jednotlivé webové služby, aniž by věděly, že jsou součástí kompozice. Orchestrace je odolnější vůči chybám, protože při selhání služby může být nahrazena jinou službou a neovlivní to celý proces, pouze jeho část.

4.3.1 Příklad využití v praxi

Choreografii si můžeme ilustrovat na práci lékaře. Lékař musí provést na sobě závislé úkony vedoucí k určení diagnózy. Příklad, kdy si pacient způsobí luxaci ramenního kloubu, můžeme popsat tímto procesem: lékař zjistí nezbytné informace od pacienta a na tomto základě určí předběžnou diagnózu. Na základě předběžné diagnózy provede vyšetření, například pomocí rentgenového zařízení. Díky údajům z vyšetření může lékař určit konečnou diagnózu a rozhodnout o následném dalším postupu a léčbě. Postup může zahrnovat dále operativní zákrok, nebo pouze fixaci kloubu.

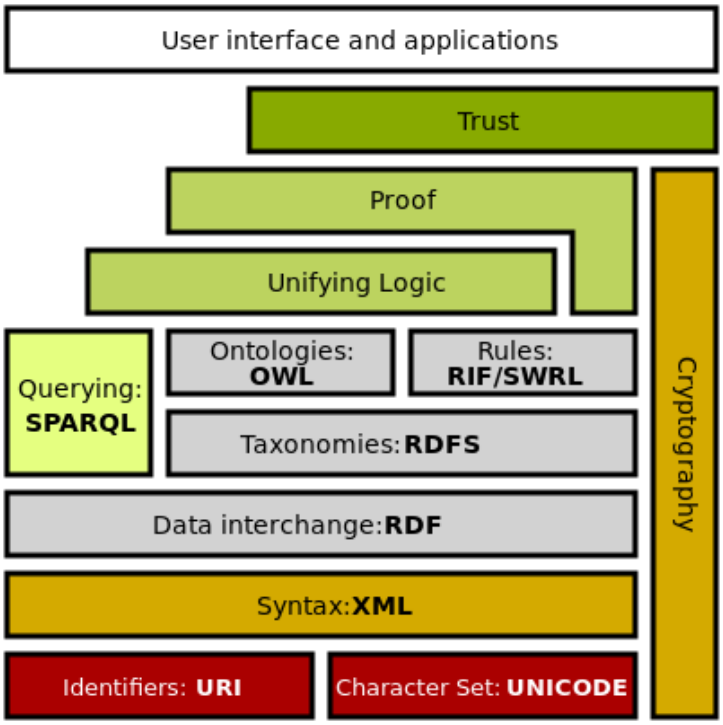
Z předešlého příkladu lze určit, že lékař neprovádí všechny aktivity sám a jeho diagnóza je závislá na předchozích krocích. Veškeré kroky jsou definovány předem daným procesem.

4.3.2 Standardy choreografie webových služeb

Standardy choreografie webových služeb se překrývají se standardy pro orchestraci. Využíváme tedy BPMN notaci pro modelování choreografie.

4.4 Sémantický web

Pro automatickou kompozici potřebujeme nadstavbu klasického webu. Klasický web je orientovaný na člověka s využitím HTTP a URI. Obsah webu je napsán přirozeným jazykem tak, aby byl co nejvíce pochopitelný pro člověka. Nadstavbu klasického webu tvoří sémantický web, který se orientuje na kooperaci strojů s člověkem. Klasickým příkladem je návštěva úřadu, kdy nemusíme znát úřední hodiny, ale na základě inteligentního osobního kalendáře se určí nejvýhodnější čas pro návštěvu spolu s automatickým objednáním návštěvy úřadu. Základními standardizovanými technologiemi sémantického webu popisující formální definici konceptu, podmínky a vztahy s doménou jsou: RDF, RDFS, SKOS, SPARQL, N3, OWL a RIF.



Obrázek 10: Ilustrace architektury Sémantického Webu [26]

4.4.1 RDF

Resource Description Framework je aplikační rámec vyjadřující informace o zdrojích. Zdrojem může být cokoliv, včetně dokumentů, lidí, fyzických objektů a abstraktních pojmů. RDF je určen pro situace, kdy je potřeba informace na webu zpracovávat aplikacemi, nikoliv pouze pro zobrazení pro lidi. RDF poskytuje rámec pro vyjádření těchto informací, což při jejich vyměňování mezi aplikacemi zabraňuje ztrátě jejich významu. Při parsování a zpracovávání mohou být využity běžně dostupné nástroje. Schopnost výměny informací mezi různými aplikacemi znamená, že informace mohou být zpřístupněny jiným aplikacím, než pro které byly původně vytvořeny [27].

RDF můžeme tedy využít k publikaci a propojení dat na Webu. Data mohou být identifikovatelná pomocí URI nebo IRI, což umožňuje tzv. propojení dat. K serializaci pro různé datové formáty můžeme využít následující:

- **Turtle** a **TriG** využívají textovou reprezentaci RDF grafu pro trojice (předmět-predikát-objekt: David má 25 let.).
- **JSON-LD** je lehká syntaxe pro serializaci propojených dat v JSON formátu.
- **RDFa** popisuje zdroje v atributech v HTML elementech stránky.
- **N-Triples** a **N-Quads** využívají lineární formát k serializaci prostého textu RDF grafu, je to podmnožinou Turtle.

4.4.2 RDFS

RDF Schéma je sémantické rozšíření RDF. Poskytuje mechanismus popisující skupiny souvisejících zdrojů a vztah mezi těmito zdroji. Systém tříd a vlastností je v RDFS podobný objektově orientovaným programovacím jazykům, jako je Java[29]. RDFS je definován tak, aby nabízel rozlišující slovník pro modelování hierarchie tříd a vlastností a dalších základních prvků schémat, které lze odvodit z RDF. RDFS v terminologii znalostního inženýrství představuje jednoduchou ontologii, díky které lze ověřit konzistence RDF dokumentů [30].

4.4.3 SKOS

Simple Knowledge Organization System je společný datový model pro systémy založených na znalostech, jako jsou tezaurus, klasifikační schémata, hesláře a taxonomie. Pomocí SKOS lze organizaci znalostního systému vyjádřit jako strojově čitelná data. Díky tomu mohou být data vyměňována mezi aplikacemi a publikována ve strojově čitelném formátu na webu [28].

4.4.4 SPARQL

SPARQL ve verzi 1.1 je sada specifikací, které poskytují jazyky a protokoly pro dotazování a manipulaci s obsahem RDF grafů na webu nebo v RDF úložišti. Specifikace se týkají: dotazovacího jazyka, formátování výsledků (např. JSON), federativního vyhledávání, rozšiřovacích režimů (definuje sémantiku SPARQL dotazů), aktualizačního jazyka, protokolu pro RDF, popisu služeb (SD), ukládání RDF grafů v HTTP operacích a testovacích případů [31].

4.4.5 N3

N3 je kompaktní a čitelná alternativa k syntaxi RDF XML. N3 umožňuje zobrazení RDF, ale zdůrazňuje čitelnost a symetrii. Umožňuje citování nebo prohlášení o prohlášeních. Toto umožňuje uživatelům rozlišovat mezi tím, co považují za pravdivé, co někdo jiný, včetně webové stránky, uvádí, nebo tomu věří. N3Logic využívá N3 syntaxi a rozšiřuje RDF slovníkem predikátů. N3 cílí na logické informace o tom, co RDF dělá pro data: poskytnout společný datový model a společnou syntaxi tak, aby rozšíření jazyka bylo snadno provedeno definováním nových termínů v ontologii [32].

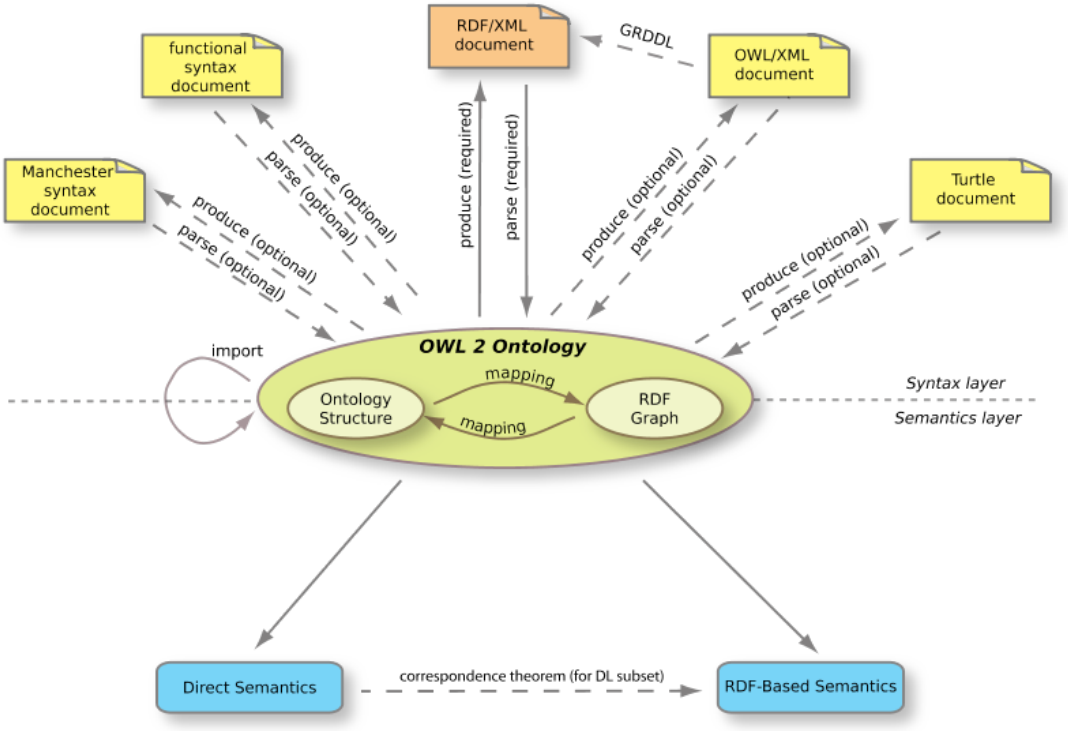
4.4.6 RIF

Skupina Rule Interchange Format v rámci konsorcia W3C si kladla za cíl vytvořit standard pro výměnu pravidel mezi různými systémy, zejména systémů pracujících na Sémantickém Webu. Vzhledem k velkému množství, jak komerčních, tak i výzkumných prototypů s velkou škálou vlastností, které se liší syntakticky i sémanticky, nebyl tento cíl jednoduchý. RIF je sbírka dialektů, což je rozšiřitelná sada jazyků s přísně definovanou syntaxí a sémantikou. Rozšiřitelnost zde znamená, že mohou být přidávány nové dialekty. Myšlenkou výměny pravidel prostřednictvím RIF je, že různé systémy jsou schopny mapovat své jazyky, nebo jejich podstatné části, z a do příslušných dialektů RIF [33].

4.4.7 OWL

OWL je Sémantický Webový jazyk navržený tak, aby reprezentoval bohaté a komplexní znalosti o věcech, skupinách a vztazích mezi věcmi. OWL je počítačový jazyk založený na deskripční logice[34] tak, že znalosti vyjádřené OWL mohou být využívány počítačovými programy (např. verifikace konzistence znalostí nebo přetvoření implicitních znalostí na explicitní). Dokumenty OWL, známé jako ontologie, mohou být publikovány na webu, nebo mohou být odkázány z jiných OWL. Následující obrázek uvádí přehled jazyka OWL 2. Elipsa symbolizuje abstraktní pojem ontologie (může vyjadřovat abstraktní strukturu nebo RDF graf). Nad elipsou jsou konkrétní syntaxe, které slouží k serializaci a výměně ontologií. Ve spodní části jsou dvě sémantické specifikace definující význam OWL. Většina uživatelů využívá pouze jednu syntaxi a jednu sémantiku.

4.4.7.1 Ontologie Ontologie je explicitní specifikace konceptualizace. Termín je vypůjčený z filozofie, kde je ontologie systematickým popisem Existence. Definice spojují názvy entit v univerzu diskurzu (např. třídy, vztahy, funkce nebo jiné objekty) s člověkem čitelnou formou, a tím definují formální axiomy, které omezují interpretaci a zaručují správné použití [35]. Ontologie se využívají v umělé inteligenci, Sémantickém Webu, softwarovém inženýrství apod. V těchto disciplínách definují základní datové modely, které reprezentují určitou znalost, nebo její část.



Obrázek 11: Struktura OWL 2 [34]

4.4.8 Sémantická anotace

Základním prvkem Sémantického Webu je zpracování zdrojů. Tyto zdroje musí být strojově zpracovatelné a unifikované. Identifikátory (URI) můžeme přirovnat k atomům a sémantické anotace k molekulám. Sémantické anotace vytváří vztah mezi identifikátory a buduje sítě dat. Anotace připojují k datům údaje (metadata) z jiných zdrojů, které následně slouží pro zpracování. Anotace můžeme dělit na tři typy [36].

1. Neformální
2. Formální
3. Ontologické

Neformální nejsou strojově čitelné, protože nepoužívají formální jazyk. Formální jsou strojově čitelné, ale nevyužívají ontologické pojmy. Ontologická anotace má obecně pochopitelný význam, který odpovídá konceptualizaci ontologie [36].

Anotace jsou typicky určeny pro využití lidmi nebo zprostředkovateli (agenty). Anotace webových zdrojů se liší od anotace webových služeb [37].

4.4.8.1 Proces anotace webových zdrojů Webové zdroje využívají k anotaci model reálného světa. Model reálného světa je popsán ontologickým schématem, který poskytuje dohodnutý a jednoznačný model pro zachycení dat a metadat. Dále využívá znalostní základ, což jsou instance ontologie. Následuje proces identifikace entity, který zahrnuje lexikální analýzu, zpracování přirozeného jazyka, lingvistické zdroje, tezaurus a slovníky. Dále je v procesu rozcestník pro odstraňování dvojího pochopení. Posledním krokem je samotná anotace. Tyto tři hlavní kroky se liší v závislosti na druhu zdroje, který je anotován [37].

4.4.8.2 Proces anotace webových služeb Proces anotace webových služeb spočívá v definování jasné sémantiky dat a funkcionality využívaných danou webovou službou. Jednotlivé prvky webové služby jsou označeny v doménových modelech nebo ontologiích, což vylučuje jakoukoliv nejednoznačnost při interpretaci funkcionality nebo dat, a tím umožňuje automatizaci objevování a kompozice služeb.

Nejednoznačnost může být znázorněna na dvou webových službách, které pracují naprosto s jiným účelem, ale využívají stejné typy dat, názvy operací a vstupy/výstupy. Pro zajištění anotace musíme tedy znát přesnou sémantiku.

5 Trend a předpoklad vývoje kompozitních služeb

V posledních letech se veškerý software začal přetvářet na webové služby, a proto jsou na webové služby kladeny vyšší nároky. Nejnovější zákaznická řešení vyžadují řešení pomocí SaaS. Díky této technice je zdokonalována využitelnost webových služeb. Motivace společností poskytovat služby, které jsou dostatečně obecné, je vyšší z ekonomických důvodů, díky reciprocitě investic. Společnosti dodávají potřebné zdroje ke zdokonalení existujících řešení za pomoci služeb k co nejvyšší efektivnosti. Tato část práce se zabývá současnými trendy ve vylepšování webových služeb. Ačkoliv jsou některá vylepšení stará i desítky let, tak se začala uplatňovat plně až v dnešní době.

5.1 Zdokonalování webových služeb

Existuje velké množství zdokonalení služeb od komunikačních po implementační. Uvedeno bude několik nejvýznamnějších.

5.1.1 REST API

Prvním, ale důležitým vylepšením, je zdokonalování samotných webových služeb. Dnes je již standardem komunikace aplikací prostřednictvím REST API, což zajišťuje možnost komplexní komunikace skrze síťové protokoly (nejčastěji HTTP).

5.1.2 SOA

Dalším vylepšením je koncept SOA. Jak již bylo v předchozí sekci napsáno, tak s rostoucím vývojem webových služeb je nutné upravit i architekturu softwaru. Dnes se největší procento softwaru vytváří na architektuře SOA, právě z důvodu nabízení SaaS.

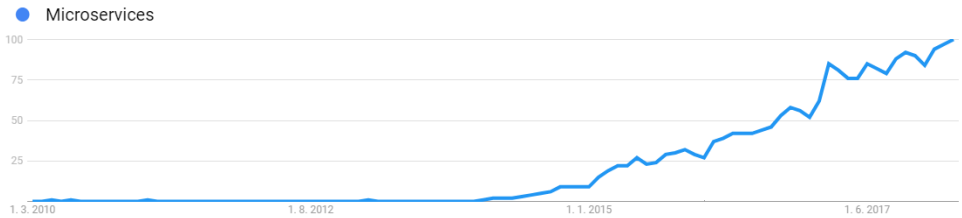
5.1.3 Microservices architektura

Každá webová služba by měla být zaměřena na určitou činnost. Tato činnost se může skládat z velkého množství kroků, které daná služba vykonává. Myšlenka Microservices posouvá tuto problematiku obecnějším směrem. V rámci dané služby může být vykonáváno několik dalších služeb, které jsou zde zapouzdřeny a pro okolí skryty. Komunikace mezi jednotlivými službami probíhá skrze jejich API. Microservice architektura popisuje konkrétní způsob navrhování softwarových aplikací jako sady nezávislých nasaditelných služeb [38].

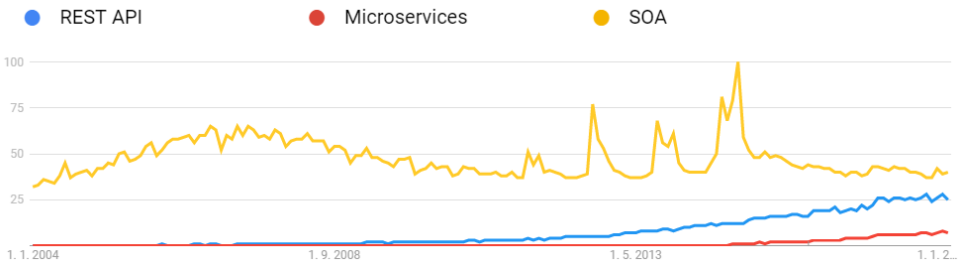
5.1.4 Srovnání

Trend Microservices ukazuje i statistika z Googlu, která ukazuje zájem o téma v průběhu času od roku 2010 do současnosti.

Další graf ukazuje závislosti všech třech zmíněných zdokonalení. Lze vidět, že zájem v průběhu času je nejvyšší u SOA. Zájem o REST API v posledních letech poměrně rychle roste a sílí zájem taktéž o Microservices, avšak Microservices je z nich nejmladší.



Obrázek 12: Trend Microservices od roku 2010 [39]



Obrázek 13: Trend REST API, SOA a Microservice od roku 2004 [40]

5.2 Vylepšení anotace služeb, automatický popis

Jedním z předpokládaného dalšího vývoje automatické kompozice služeb je automatický (ideální i sémantický) popis služby (SD). Na základě vstupů a výstupů [41] je možné identifikovat typ webové služby. Důležitým krokem k této automatizaci je automatizovat i sémantiku, což je velmi složitá procedura. Předpokladem vytvoření vhodné struktury SD je využití ontologií, tedy udělat správný popis funkcionality služby. Pakliže bychom provedli všechny tyto úkony správně, bylo by možné využít nějaký ontologický jazyk spolu se slovníkem k extrakci klíčových bloků dané služby. Zde se můžeme bavit o automatické sémantické anotaci, kdy mohou být díky identifikaci důležitých parametrů, přiřazena metadata pro danou službu, a tím zaručit její budoucí využití.

5.3 Umělá inteligence při kompozici založené na QoS

Plánování a genetické algoritmy byly nejrozšířenější techniky umělé inteligence pro automatizaci kompozice webových služeb. Problémy spojené s kompatibilitou a přenositelností webových služeb vyžadují výzkum k efektivnímu komponování služeb v heterogenních a decentralizovaných prostředích [42]. Umělá inteligence představuje budoucnost kompozice webových služeb, a nejenom těch. Techniky, jako rojové algoritmy, se začínají využívat napříč celým softwarovým světem (např. počítačové viry). Rojové algoritmy jsou biologicky inspirované algoritmy, které cílí na vyhledávání optimální cesty.

5.4 Využití Sémantického Webu

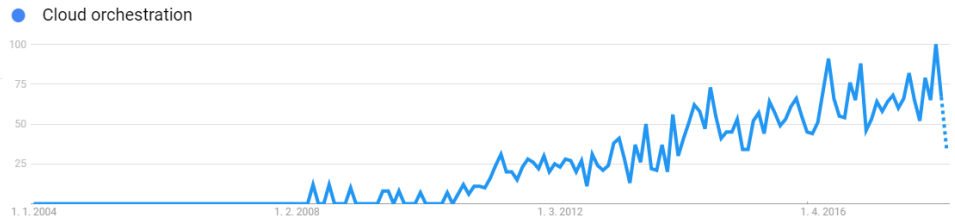
Využití Sémantického Webu není současným směrem v kompozici webových služeb, avšak je nutné podotknout, že potenciál využití Sémantického Webu pořád existuje. V kompozici velké množství řešení využívá ontologie, což je základní prvek Sémantického Webu. Je nutné také doplnit, že existuje i velké množství těch, které nevyužívají žádnou ontologii. To je možná jeden z důvodů, proč se zatím Sémantický Web neuchytil mezi standardními řešeními. Paradoxní na těchto faktech je to, že velká softwarová řešení jej využívají, například společnost Google využívá technologie podobné Sémantickému Webu.

5.5 Cloudová orchestrace

Cloudová orchestrace je použití programovací technologie pro správu propojení a interakcí mezi pracovní zátěží na veřejné nebo privátní cloudové infrastruktuře. Spojuje automatizované úkoly do soudržných pracovních postupů k dosažení cíle s kontrolou oprávnění a vynucené politiky. Cloudová orchestrace se typicky využívá k poskytování, nasazování nebo spouštění serverů, získávání a přiřazování úložiště, spravování sítě, vytváření virtuálních strojů a k přístupu k určité službě skrze cloud [43].

Cloudová orchestrace je E2E automatizace zavádění služeb v prostředí cloudu. Konkrétně jde o automatizované uspořádání, koordinaci a správu složitých počítačových systémů, middlewaru a služeb, což přispívá k urychlení poskytování IT služeb při současném snížení nákladů. Pomocí vhodných orchestračních mechanismů mohou uživatele nasadit a začít používat služby na serverech nebo na libovolných cloudových platformách. Existují tři aspekty cloudové orchestrace: orchestrace zdrojů, orchestrace pracovních postupů a orchestrace služeb [44]. Zjednodušeně lze říci, že cloudová orchestrace zobecňuje a zjednodušuje automatizaci uspořádání úkolů na konkretních strojích a zejména tam, kde existují různé vnitřní i vnější závislosti.

V předchozím textu lze zaznamenat, že orchestrace se provádí dnes téměř na všem. Předpokládaný vývoj další automatizované kompozice služeb na těchto platformách nelze určit přesně, jelikož dynamika změn cloudových služeb, jako celku, je nepředvídatelná. Lze ale předpokládat, že klasické poskytování softwaru (včetně infrastruktury) klientům skončí a přesune se kompletně do cloudů. Cloud nabízí větší možnosti škálovatelnosti a snadné integrace. Na dalším obrázku lze vidět trend cloudové orchestrace.



Obrázek 14: Trend Cloudové orchestrace od roku 2004 [45]

Výběr poskytovatelů infrastruktury a nástrojů pro cloudovou orchestraci:

- Amazon Web Services CloudFormation
- Cisco CloudCenter
- CloudFX
- IBM Cloud Orchestrator
- Microsoft Azure Automation
- Microsoft Cycle Computing
- OpenStack Heat orchestration engine
- Saltstack
- RightScale
- Cloudify
- Chef
- Puppet
- Docker

6 Prototypové řešení

Automatická kompozice služeb zahrnuje velké množství architektonických a technologických přístupů. Každý tento přístup má svou specifickou implementaci a potýká se různými problémy. Prototypové řešení této práce se bude zaobírat poloautomatickou kompozicí RESTful webových služeb. Je nutné zdůraznit, že prototypové řešení si neklade za cíl naprostou správnost přístupu, ale snaží se navrhnout koncept, podle kterého by mohlo být implementováno produkční řešení. Koncept využívá triviální proces objednání cesty prostřednictvím RESTful webových služeb. Řešení je postaveno na Java platformě a klade si za cíl využít co nejvíce nových přístupů dané platformy.

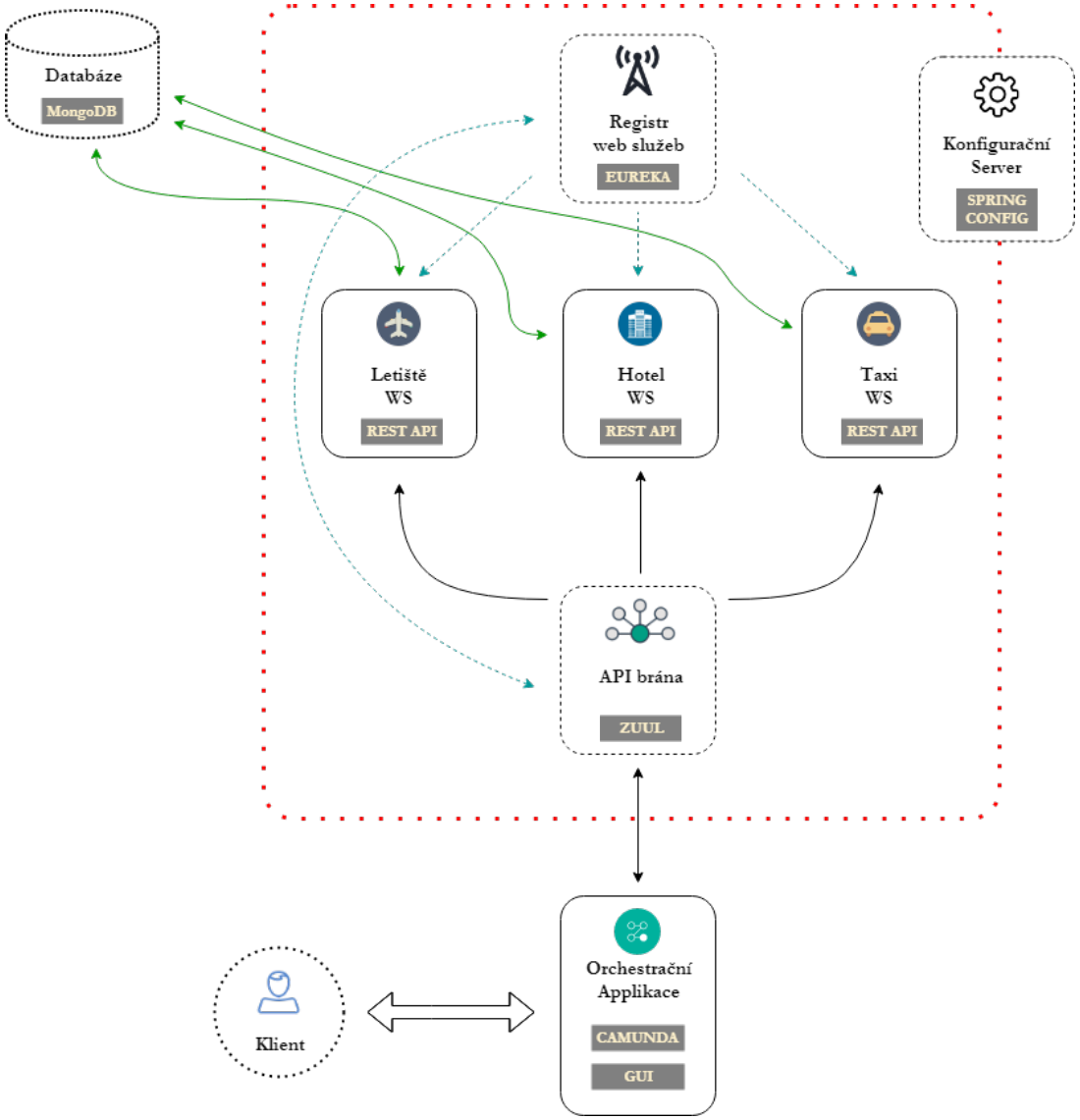
6.1 Vize a požadavky na koncept

V počátcích návrhu prototypového řešení této práce byla vize orientována spíše na plně automatickou kompozici. Po důkladném prostudování a posouzení řešení bylo určeno, že plně automatická kompozice služeb je obtížná na implementaci (např. z důvodů správných ontologií a sémantického znázornění RESTful webových služeb). Proto bylo vybráno řešení, které se začíná více prosazovat do praxe. Toto řešení je poloautomatická kompozice RESTful webových služeb. Tato kompozice je zpracována podnikovým procesem, který je popsán notací BPMN.

Požadavkem na prototypové řešení je nezávislost webových služeb. To je docíleno prostřednictvím architektury microservices. Ta umožňuje jednotlivé dílčí úkony podnikového procesu rozdělit na logické celky. Pro každý logický celek je vytvořená zvláštní RESTful webová služba. Dalším požadavkem je perzistence výsledků zpracování procesů, což nabízí využití jiného, než relačního databázového systému.

6.2 Model navržené architektury vytvářené platformy

Model navržené architektury vytvářené platformy znázorňuje veškeré logické celky konceptu. Na obrázku lze zaznamenat různé typy vazeb a objektů. **Letiště WS**, **Hotel WS** a **Taxi WS** jsou RESTful webové služby implementované autorem řešení. Další kompletně implementovanou částí je **Orchestrační Aplikace**. Z obrázku lze vypozorovat také využití **databáze**, se kterou pracují dané webové služby. Dále jsou v modelu **Registr web služeb** a **API brána**, kde první z nich registruje veškeré webové služby a uchovává informace o jejich stavu. Brána centralizuje přístup k jednotlivým službám, takže není nutné přistupovat ke službám odděleně. Nad všemi službami, registrem a bránou stojí **Konfigurační server**, který poskytuje centralizované řízení konfigurací, jako nastavení portů databáze, registru služeb apod.



Obrázek 15: Model navržené architektury vytvářené platformy

6.3 Popis implementace

Tato část práce popisuje důležité části prototypového řešení, které jsou nezbytné k ilustraci poloautomatické kompozice. Prototypové řešení je dostupné v repozitáři `git.cs.vsb.cz`[46].

6.3.1 Kostra aplikace a definované konfigurace

Kostra prototypového řešení byla vložena do nově založeného projektu v GitLabu. Tato kostra obsahuje Maven projekt s dalšími Maven podprojekty, což jsou jednotlivé funkční celky prototypu (webové služby, registr, konfigurační server, brána a orchestrační aplikace). Veškeré funkční celky byly vytvořeny pomocí nástroje Spring Boot, který umožňuje snadné a rychlé vytvoření webové aplikace na Java platformě s využitím Spring frameworku. V modelu navržené architektury lze najít **Konfigurační server**, který obsahuje veškerou konfiguraci pro větší část prototypu. Tyto konfigurace se nachází v souborech s příponou `.yml`. V těchto souborech jsou konfigurace databáze, portů apod. (viz ukázka .yml souboru).

```
server:
  context-path: /airplane
  port: 24300
spring:
  data:
    mongodb:
      host: mongodb
      database: asc
      port: 27017
```

Výpis 5: Ukázka YAML souboru pro webovou službu

6.3.2 Aplikační logika služeb a unit testy

Komunikace všech implementovaných služeb probíhá skrze REST API. Díky použití Spring frameworku je snadné takovéto API vytvořit (viz ukázka kódu). Aplikace má tři vrstvy: komunikační (REST API), aplikační (logika) a perzistentní (komunikace s databází). Aplikační vrstva zaobaluje chování prostřednictvím servisní vrstvy. Vrstva perzistence je pouze java rozhraní, kterým se přistupuje do databáze.

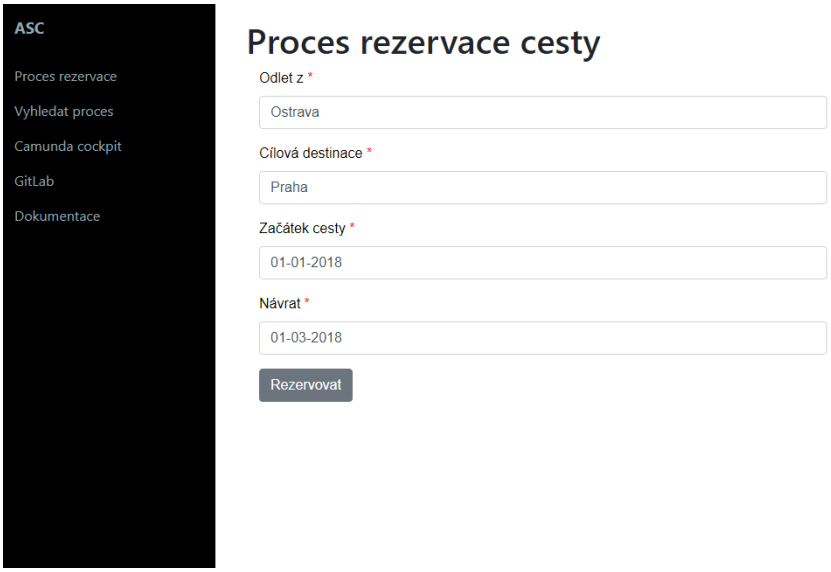
```
@RestController
@RequestMapping("/reservations")
public class AirplaneReservationServiceController {
   ...
    @Autowired
    private AirplaneReservationService airplaneReservationService;

    @GetMapping("/{uuid}")
    public ResponseEntity<?> get(@PathVariable String uuid) {
      logger.debug("{AirplaneReservationServiceController-get} - Getting
          airplane reservation");
      AirplaneReservation airplaneReservation = airplaneReservationService.get(
          uuid);
      return new ResponseEntity<>(airplaneReservation, HttpStatus.OK);
    }

    @PutMapping
    public ResponseEntity<?> create(@Valid @RequestBody String data) {
      logger.debug("{AirplaneReservationServiceController-create} - Creating
          airplane reservation");
      AirplaneReservation airplaneReservation = airplaneReservationService.
          create(data);
      return new ResponseEntity<>(airplaneReservation, HttpStatus.OK);
    }
    ...
```

Výpis 6: Ukázka REST API

Každá softwarová část by měla být patřičně otestována, avšak prototypové řešení ve většině případů nezahrnuje unitové testy na všechny stavy softwaru. V řešení jsou tedy implementovány testy úspěšných scénářů a několik možných neúspěšných. Vše je testováno za pomocí JUnit a Mockito frameworků, které usnadňují automatizaci testování a mockování služeb. Testována je veškerá funkčnost, včetně HTTP požadavků.

6.3.3 Orchestrační aplikace

Další hlavní částí implementace prototypu je orchestrační aplikace. Tato aplikace nabízí jednoduché uživatelské rozhraní pro práci. GUI je implementováno pomocí HTML stránek za použití Thymeleafu, Spring frameworku, JS, Bootstrapu a CSS. Spring framework využívá MVC architektonický styl, což logicky odděluje HTML stránky od funkcionality. Pro aplikaci byl navržen pouze jeden hlavní `Controller`, který manipuluje s modelem. Tento controller využívá servisní vrstvu aplikace. Implementace servisní vrstvy zaobaluje funkcionalitu Camunda softwaru, což nebylo nutné, jelikož Camunda obsahuje REST API pro práci s ním. Servisní vrstva tak byla vytvořena pouze pro přehlednost a zapouzdření základních ovládacích prvků Camundy.



Obrázek 16: Proces nasazený v prototypu

Samotná Camunda je klíčovou části orchestrační aplikace. Skrze Camundu se spouští jednotlivé podnikové procesy, které jsou vytvářeny pomocí BPMN nástrojů. Camunda rovněž obsahuje své GUI, které se jmenuje Camunda Cockpit. Orchestrační aplikace obsahuje odkaz na toto GUI. V Cockpitu lze vidět veškeré nasazené procesy, spravovat tyto procesy, provádět úkony apod.

Aplikaci není snadné otestovat z pohledu správné funkcionality. Proto nebyly implementovávaný žádné unitové testy.

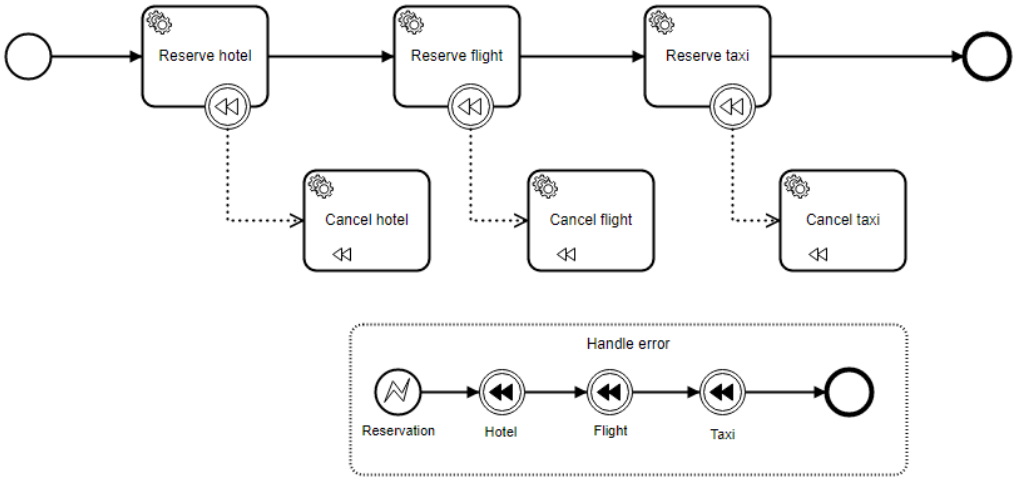
6.3.4 Podnikový proces

Camunda využívá pro svou práci BPMN procesy (zvládá i jiné např. DMN). V prototypovém řešení byl navržen proces rezervace cesty. Veškeré jednotlivé pracovní jednotky (`tasky`) BPMN procesu jsou `servisní` a využívají `http-connector`, který není implicitně zahrnut v Camundě [47].

Každý konektor servisní jednotky obsahuje parametr `headers`, který definuje akceptovaný formát dat (`application/json`). Dále obsahuje parametr `method` znázorňující HTTP metodu požadavku. Odkaz na externí službu je obsažen v parametru `url`. Posledním vstupním parametrem je `payload`, do kterého jsou uloženy vstupní parametry služby.

Výstupním parametrem je hodnota zpracována jednotkou. Tato hodnota je obsažena v HTTP odpovědi, takže musí být zpracována například skriptem.

Dalšími důležitými součástmi procesu jsou kompenzační stavy. Kompenzační stav definuje co se stane, jakmile dojde v dané jednotce k chybě. V bloku zpracování chyby lze vypozorovat posloupnost úkonů při chybovém stavu aplikace.



Obrázek 17: Proces rezervace cesty

6.3.5 Využití Dockeru

Celý prototyp využívá Docker, který zjednodušuje práci s virtuálními kontejnery. Docker je ideální pro testování a nasazení softwarů založených na microservices architektuře. Dokáže snadno oddělit hostitelský stroj od testovaného kontejneru.

V prototypu má každá webová služba, konfigurační server, registrační server a brána, svůj vlastní kontejner, což simuluje reálné použití. Každá služba má svůj `Dockerfile`, kde se definují potřebné informace k vytváření kontejneru.

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ARG JAR_FILE
ADD ${JAR_FILE} app.jar
EXPOSE 24300
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Výpis 7: Ukázka Docker souboru

6.3.5.1 MongoDB

Snadnou instalaci MongoDB lze provést prostřednictvím Dockeru. Není tak nutné implementovat další aplikaci, která má za úkol instalovat databázi na serveru. MongoDB je spuštěno v Dockeru na portu `27017`.

6.3.5.2 Docker compose

Docker compose je nástroj, který lze využít pro automatické nasazení více kontejnerů zároveň. Docker compose zvládá i další pokročilou funkcionalitu. V prototypu je díky tomuto nástroji snadno instalována instance MongoDB do kontejneru bez nutnosti další konfigurace. V prototypu je taktéž nasazen konfigurační server a ostatní závislé softwarové celky jsou závislé na úspěšném nastartování serveru.

```
version: '3'
services:
  config-service:
    image: asc/config-service:latest
    ...
    healthcheck:
      test: curl --fail -s http://config-service:8888/actuator/health || exit 1
      ...
  airplane-service:
    image: asc/airplane-service:latest
    ...
  mongodb:
    image: mongo:latest
    container_name: "mongodb"
    ...
    ports:
        - 27017:27017
  ...
```

Výpis 8: Ukázka Docker compose souboru

6.3.6 Sestavení a nasazení

Sestavování aplikace probíhá pomocí Mavenu. Pro úspěšné sestavení prototypu Spring boot pro Docker je nutné využít Maven Wrapper [48].

K nasazování prototypu je určen Docker compose, který lze následně propojit s poskytovateli cloudové infrastruktury.

6.3.7 Souhrn implementovaných služeb a aplikací

Souhrn veškerých implementovaných webových služeb a aplikací.

6.3.7.1 Konfigurační služba (Cloud Config server) Konfigurační služba (server) poskytuje centralizovaný přístup ke konfiguracím. Jedná se o reprezentaci konfiguračního managementu pro další služby a celky. V prototypu je této službě přiřazen port `8888`. Využívá YAML popřípadě `.properties` souborů.

6.3.7.2 Služba pro automatické vyhledávání služeb (Eureka server) Služba pro automatické vyhledávání služeb je určena pro registraci webových služeb, které jsou dostupné v rámci dané sítě. Tato služba reprezentuje instanci Eureka serveru od Netflix OSS. V prototypu je této službě přiřazen port `8761`. Jednotliví klienti jsou připojováni automaticky prostřednictvím anotace `@EnableDiscoveryClient`.

6.3.7.3 Služba brány (Gateway server) Služba brány v prototypu poskytuje centralizovaný přístup k jednotlivým službám, které jsou zaregistrovány pomocí Eureka serveru. Efektivně a dynamicky přidává cesty na dané služby. V prototypu je této službě přiřazen port `8765` a využívá implicitní cesty ke službám (např. ke službě taxi se připojujeme jejím jménem taxi-service).

6.3.7.4 Služba pro rezervaci letu Služba pro rezervaci letu umožňuje klientovi této služby spravovat rezervace letu letadlem. V prototypu je této službě přiřazen port `24300`. Poskytuje RESTful API pro práci s danou službou.

6.3.7.5 Služba pro rezervaci hotelu Totožná služba jako pro rezervaci letu běžící na portu `24200`.

6.3.7.6 Služba pro rezervaci taxi Totožná služba jako pro rezervaci letu běžící na portu `24100`.

6.3.7.7 Aplikace pro automatizaci podnikového procesu Aplikace obsahující GUI, které je vystaveno klientovi. Klientovi je umožněno spouštět proces rezervace a sledovat výsledky. V prototypu je této službě přiřazen port `8080`. Využívá Camunda software pro správu podnikových procesů.

6.3.8 Souhrn použitých technologií a nástrojů

6.3.8.1 Spring framework Spring usnadňuje vytváření podnikových aplikací založených na Javě. Poskytuje vše potřebné pro podnikové prostředí, podporuje Groovy a Kotlin jako alternativní jazyky pro JVM. Umožňuje flexibilně vytvářet mnoho druhů architektur v závislosti na potřebách aplikace [49].

6.3.8.2 Spring Boot Umožňuje snadné vytváření samostatných produkčních aplikací, založených na Springu frameworku, které lze spouštět. Spring aplikaci lze založit s minimálním úsilím. Většina Spring Boot aplikací vyžaduje velmi malé změny konfigurace [50].

6.3.8.3 Camunda Camunda BPM je lehká otevřená platforma pro řízení podnikových procesů [51].

6.3.8.4 Docker Docker je platforma pro vývojáře a systémové administrátory pro vývoj, nasazení a spouštění aplikací prostřednictvím kontejnerů. Docker umožňuje skutečnou nezávislost mezi aplikacemi, infrastrukturou a vývojáři pro zvýšení potenciálu a vytvoření modelu pro lepší spolupráci a inovativnosti [52].

6.3.8.5 Docker compose Docker Compose je nástroj pro definování a provoz většího množství kontejnerů Dockeru. Využívá YAML soubory ke konfiguraci služeb aplikace. Jediným příkazem vytváří a spouští všechny služby z konfigurace [53].

6.3.8.6 MongoDB MongoDB je databáze dokumentů se škálovatelností a flexibilitou, která je potřebná při dotazování a indexování [54].

6.3.8.7 Thymeleaf Thymeleaf je šablonový systém na straně Java serveru určený pro webové i samostatně stojící prostředí [55].

6.3.8.8 JUnit JUnit je jednoduchý framework s otevřeným kódem pro psaní a spouštění opakovatelných testů [56].

6.3.8.9 Mockito Mockito je mockovací framework pro unitové testy v Javě [57].

6.3.8.10 Maven Maven je software pro řízení a porozumění projektu. Na základě konceptu POM může Maven centralizovaně spravovat sestavování, reportování a dokumentaci informace o projektu [58].

6.3.8.11 Maven Wrapper Maven Wrapper je nástroj zajišťující vše potřebné pro spuštění Maven sestavení [59].

6.4 Výsledky experimentů a popis konceptu

Nejdůležitějším výsledkem experimentů je potvrzení správnosti konceptu poloautomatické kompozice RESTful webových služeb (založených na architektuře microservices) prostřednictvím BPMN definovaných procesů. Správná funkčnost jednotlivých RESTful webových služeb dokazuje skutečnost, že nově vyvíjená softwarová díla mohou být poloautomaticky komponována i na cloudových platformách.

Dalším experimentem, v rámci konceptu, bylo využití konfiguračního serveru, který poskytuje potřebné konfigurace dalším potřebným celkům prototypu. Prokázalo se, že nasazení konfiguračního serveru je vhodné tam, kde je velký počet nabízených webových služeb, které mají podobné nastavení. Jediným menším problémem je skutečnost, že konfigurační server musí být vždy dostupný, což při velkém zatížení sítě nemusí být možné. Toto je ale problém všech centralizovaných řešení. Řešením tohoto problému je automatické vyvažování zatížení, což Spring framework umožňuje, ale prototyp tento princip nevyužívá.

Klíčový experiment proběhl s notací BPMN. Experiment měl dokázat použitelnost BPMN notace v prostředí cloudové infrastruktury. Řešení orchestrace webových služeb pomocí BPMN notace je proveditelné, avšak má svá úskalí například v asynchronním volání jednotlivých služeb. Camunda umožňuje snadnou implementaci takového řešení.

Experimentování se Spring frameworkem ukázalo, že Spring framework (konkrétně Spring Boot) je velmi vhodný pro architekturu microservices spolu s nástrojem Docker.

6.5 Zhodnocení řešení a popis návrhu dalšího rozvoje konceptu

Základním pilířem prototypového řešení jsou RESTful webové služby. Tyto služby jsou postaveny na microservices architektuře, což dává daným službám požadovanou výkonnost a zvyšuje jejich nezávislost (část 5.1.3). Cílem práce bylo navržení a implementace automatické kompozice webových služeb. Plně automatická kompozice je nesmírně složitá na implementaci, a proto bylo vybráno řešení poloautomatickou kompozicí (část 3.2.2). Poloautomatická kompozice je v řešení založena na definovaném podnikovém procesu (část 6.3.4) v notaci BPMN. Proces popisuje činnost vedoucí k objednání cesty prostřednictvím letecké společnosti, hotelové a taxi služby. Řešení využívá prvek pro řízení konfigurací (část 6.3.7.1), což umožňuje centralizovat nastavení klíčových prvků prototypu. Dále prototyp obsahuje API bránu (část 6.3.7.3), přes kterou prochází veškerá komunikace s webovými službami. Tato brána využívá registračního serveru (část 6.3.7.2), který obsahuje informace o všech dostupných webových službách v dané síti. Pomocí komponování a kontejnerizace je každá daná část řešení nasazena automaticky do běhového prostředí a je schopna okamžité práce. Řešení nabízí jednoduché GUI, pomocí kterého můžeme spouštět daný proces a díky němuž vidíme výsledky v JSON formátu. Toto GUI je grafickou nadstavbou orchestrační aplikace, který provádí daný podnikový proces. Celý proces je zaznamenán do databáze.

6.5.1 Popis návrhu dalšího rozvoje konceptu

Prototypové řešení má velký potenciál pro další rozvoj. Prvním rozšířením by mohlo být přidání sémantického SD webovým službám, což by umožňovalo automatickou detekci vhodné webové služby podle určitých pravidel (například podle QoS - část 2.7). Dalším rozvojem je abstrahování webových služeb do obecných webových služeb, ze kterých následně dílčí služby zdědí obecná pravidla, což by usnadnilo a zkvalitnilo výběr služeb do procesu. Automatická detekce nejvhodnějších služeb by mohla být prováděna na základě AI (část 5.3). Rozšíření nabízeného GUI je taktéž možný rozvoj řešení, jelikož dosavadní GUI je zjednodušené a vytvořené pouze pro jeden příkladný proces. Posledním větším vylepšením by bylo využití některých dostupných RESTful webových služeb, například služby předpovědi počasí, která by navrhovala, podle typu cesty, termín se schůdným počasím.

7 Závěr

Cílem teoretické části této práce je vyhodnotit současný stav kompozice webových služeb. Práce zahrnuje vysvětlení důležitých souvisejících pojmů, jako jsou: komunikační protokol SOAP, REST architektonický styl, servisně orientovaná architektura, kvalitativní a bezpečnostní požadavky kladené na služby. Samotná kompozice se dělí na statickou a dynamickou. Tato práce se soustředí na dynamickou kompozici, která probíhá v době běhu. Dynamická kompozice je základem automatické kompozice, která se dále dělí na dva nejpoužívanější styly řízení: orchestraci a choreografii.

V práci jsou srovnány aktuální druhy kompozic (manuální, poloautomatická a automatická). Průzkum se zaměřuje výhradně na poloautomatickou a automatickou kompozici, kde poloautomatická je současným trendem. Poloautomatická kompozice je dána předem danou posloupností definovaných kroků pomocí podnikových procesů. Automatická kompozice se soustředí více na sémantiku daných webových služeb a je schopna automaticky vybrat ideální webovou službu na základě parametrů (například vstupů a výstupů, kvality služeb apod.). Práce uvádí v současnosti nejpoužívanější nástroje pro BPMN notaci.

Praktická část této práce se nezaměřuje na automatickou kompozici z důvodu složitosti sémantiky, ale soustředí se na poloautomatickou kompozici RESTful webových služeb (založených na microservices architektuře) prostřednictvím podnikového BPMN procesu. Cílem prototypového řešení bylo navrhnout a implementovat koncept poloautomatické kompozice s využitím kontejnerizace. Návrh a implementace naznačují koncept využití BPMN procesů v praxi, což potvrzuje, že současná cloudová infrastruktura bude nadále rozvíjena podobnou orchestrací služeb.

Prototypové řešení obsahuje kontejnery jednotlivých částí softwaru, včetně hlavní orchestrační aplikace. Dalším přínosem řešení jsou experimenty se samostatně stojící aplikací, která zahrnuje pouze BPMN proces. Tento proces řídí orchestraci RESTful webových služeb, které mohou být dostupné odkudkoliv v dané síti.

Dalším vývojem prototypového řešení může být docíleno sémantické anotace webových služeb, což by umožnilo vývoj automatického orchestračního rámce na základě SD, a tím by se prototyp stal nezávislým na konkrétních implementacích služeb, ale využíval by abstraktních služeb. Toto možné řešení ovšem vyžaduje další výzkum v oblasti sémantiky RESTful webových služeb.

Literatura

[1] FEUERLICHT, George a Shyam GOVARDHAN. SOA: Trends and Directions [online]. [cit. 2018-04-04]. Dostupné z: https://www.researchgate.net/publication/228859308_SOA_Trends_and_Directions

[2] Web Services Architecture: Introduction [online]. 2004 [cit. 2018-01-10]. Dostupné z: https://www.w3.org/TR/ws-arch/

[3] UDDI 101: UDDI in a Web Services World [online]. 2006 [cit. 2018-01-10]. Dostupné z: http://uddi.xml.org/uddi-101

[4] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language: Introduction [online]. 2007 [cit. 2018-01-13]. Dostupné z: https://www.w3.org/TR/wsdl20

[5] SOAP Version 1.2 Part 0: Primer (Second Edition): Introduction [online]. 2007 [cit. 2018-01-13]. Dostupné z: https://www.w3.org/TR/soap12-part0

[6] FIELDING, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. IRVINE, 2000. DISSERTATION. UNIVERSITY OF CALIFORNIA.

[7] FIELDING, Roy Thomas. REST APIs must be hypertext-driven [online]. 2008 [cit. 2018-01-15]. Dostupné z: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[8] THE LINTHICUM GROUP. SOA meta-model [online]. In: . 2007 [cit. 2018-01-15]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/0/06/SOA_Metamodel.svg

[9] Service-Oriented Architecture – What Is SOA?: Service-Oriented Architecture [online]. [cit. 2018-01-15]. Dostupné z: http://www.opengroup.org/soa/source-book/soa/p1.htm#soa_definition

[10] BERNERS-LEE, Tim James HENDLER a Ora LASSILA. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American [online]. 2001, 2001(5) [cit. 2018-01-17]. ISSN 0036-8733. Dostupné z: https://www.scientificamerican.com/magazine/sa/2001/05-01/#article-the-semantic-web

[11] YONG-YI FANJIANG, YANG SYU, SHANG-PIN MA a JONG-YIH KUO. An Overview and Classification of Service Description Approaches in Automated Service Composition Research. IEEE, 2015, , 176 - 189. DOI: 10.1109/TSC.2015.2461538. ISSN 1939-1374.

[12] Web service description. IBM Knowledge Center [online]. [cit. 2018-02-01]. Dostupné z: https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.2.0/com.ibm.cics.ts.webservices.doc/concepts/dfhws_wsdl.html

[13] THIRUMARAN, M., P. DHAVACHELVAN, S. ABARNA a G. ARANGANAYAGI. Architecture for Evaluating Web Service QoS Parameters using Agents. 2010, 15-21. DOI: 10.5120/1470-1985.

[14] MEYSAM AHMADI OSKOOEI a SALWANI MOHD DAUD. Quality of service (QoS) model for web service selection [online]. IEEE, 2 October 2014, , 266-270 [cit. 2018-02-01]. DOI: 10.1109/I4CT.2014.6914187. Dostupné z: https://ieeexplore.ieee.org/document/6914187/

[15] NEWCOMER, Eric a Greg LOMOW. Web Services Security. Understanding SOA with Web Services [online]. Addison-Wesley, 2004, s. 313-348 [cit. 2018-02-01]. ISBN 978-0321180865. Dostupné z: https://www.it.iitb.ac.in/~madhumita/web%20services/Web%20Services%20Security%20chapter%208.pdf

[16] BARTALOS, Peter a Maria BIELIKOVA. AUTOMATIC DYNAMIC WEB SERVICE COMPOSITION: A SURVEY AND PROBLEM FORMALIZATION [online]. January 2011, , 793–827 [cit. 2018-02-05]. Dostupné z: http://www.cai.sk/ojs/index.php/cai/article/view/198/167

[17] DUSTDAR, Schahram a Mike P. PAPAZOGLOU. M.P. Services and Service Composition – An Introduction [online]. 2008, , 86-92 [cit. 2018-02-05]. Dostupné z: https://pdfs.semanticscholar.org/422b/e74a233ca10c0fd93e453da06cda22fed7a8.pdf

[18] SHENG, Quan Z., Xiaoqiang QIAO, Athanasios V. VASILAKOS, Claudia SZABO, Scott BOURNE a Xiaofei XU. Web services composition: A decade's overview [online]. October 2014, (Volume 280), 218-238 [cit. 2018-02-06]. DOI: 10.1016/j.ins.2014.04.054. ISSN 0020-0255. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0020025514005428?via%3Dihub

[19] MATJAZ, Juric B. a Sarang POORNACHANDRA. Preview in Mapt Code Files Business Process Execution Language for Web Services [online]. 2nd Edition. Packt Publishing, 2006 [cit. 2018-02-10]. ISBN 9781904811817. Dostupné z: https://www.packtpub.com/business-process-execution-language-web-services-2nd-edition/book

[20] OMG. Business Process Model and Notation (BPMN): formal/2011-01-03 [online]. In: . Version 2.0. January 2011 [cit. 2018-02-10]. DOI: Business Process Model And Notation. Dostupné z: https://www.omg.org/spec/BPMN/2.0

[21] OASIS. Web Services Business Process Execution Language Version 2.0: OASIS Standard [online]. 11 April 2007 [cit. 2018-02-10]. Dostupné z: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[22] OMG. Decision Model and Notation: formal/2016-06-01 [online]. In: . V1.1. May 2016 [cit. 2018-02-10]. Dostupné z: https://www.omg.org/spec/DMN/About-DMN/

[23] HESSE, Moritz. BPMN Tool Matrix [online]. [cit. 2018-02-15]. Dostupné z: https://bpmnmatrix.github.io/

[24] List of BPEL engines. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-02-15]. Dostupné z: https://en.wikipedia.org/wiki/List_of_BPEL_engines

[25] Comparison of Business Process Modeling Notation tools. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-02-15]. Dostupné z: https://en.wikipedia.org/wiki/Comparison_of_Business_Process_Modeling_Notation_tools

[26] Semantic Web Architecture [online]. [cit. 2018-02-18]. Dostupné z: http://obitko.com/tutorials/ontologies-semantic-web/semantic-web-architecture.html

[27] RDF 1.1 Primer: Introduction [online]. 24 June 2014 [cit. 2018-02-18]. Dostupné z: https://www.w3.org/TR/rdf11-primer/

[28] SKOS Simple Knowledge Organization System Reference: Introduction [online]. 18 August 2009 [cit. 2018-02-18]. Dostupné z: https://www.w3.org/TR/skos-reference/

[29] RDF Schema 1.1: Introduction [online]. 25 February 2014 [cit. 2018-02-18]. Dostupné z: https://www.w3.org/TR/rdf-schema/

[30] STAAB, Steffen, Michael ERDMANN a Alexander MAEDCHE. An extensible approach for Modeling Ontologies in RDF(S) [online]. 2000 [cit. 2018-02-18]. Dostupné z: https://www.semanticscholar.org/paper/An-extensible-approach-for-Modeling-Ontologies-in-Staab-Erdmann/08f1de5b2fea30a04fc025854e5a1c78cd6770bc

[31] SPARQL 1.1 Overview: Introduction [online]. 21 March 2013 [cit. 2018-02-18]. Dostupné z: https://www.w3.org/TR/sparql11-overview/

[32] BERNERS-LEE, Tim, Daniel Walter CONNOLLY, Lalana KAGAL a James HENDLER. Theory and Practice of Logic Programming: N3Logic: A logical framework for the World Wide Web [online]. December 2007 [cit. 2018-02-18]. DOI: 10.1017/S1471068407003213. Dostupné z: https://www.researchgate.net/publication/1736816_N3Logic_A_logical_framework_for_the_World_Wide_Web

[33] KRÖTZSCH, Markus a Umberto STRACCIA, ed. Web Reasoning and Rule Systems [online]. Springer, 2012 [cit. 2018-02-18]. ISBN 978-3-642-33203-6. Dostupné z: https://link.springer.com/book/10.1007%2F978-3-642-33203-6#editorsandaffiliations

[34] OWL 2 Web Ontology Language [online]. 11 December 2012 [cit. 2018-02-23]. Dostupné z: https://www.w3.org/TR/owl2-overview/

[35] GRUBER, Thomas R. A translation approach to portable ontology specifications [online]. 1993, , 199-220 [cit. 2018-02-23]. DOI: 10.1006/knac.1993.1008. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1042814383710083

[36] OREN, Eyal, Simon SCERRI, Siegfried HANDSCHUH a Michael SINTEK. What are Semantic Annotations? [online]. 2006 [cit. 2018-02-23]. Dostupné z: http://www.siegfried-handschuh.net/pub/2006/whatissemannot2006.pdf

[37] SINTEK, Michael a Meenakshi NAGARAJAN. What are Semantic Annotations? [online]. January 2006, , 35-61 [cit. 2018-02-23]. DOI: 10.1007/978-0-387-34685-4_2. Dostupné z: https://www.researchgate.net/publication/225859409_Semantic_Annotations_in_Web_Services

[38] FOWLER, Martin a James LEWIS. Microservices: a definition of this new architectural term [online]. 25 March 2014 [cit. 2018-02-24]. Dostupné z: https://martinfowler.com/articles/microservices.html

[39] Google Trends - Microservices [online]. In: . [cit. 2018-02-24]. Dostupné z: https://trends.google.cz/trends/explore?date=2010-02-05%202018-03-05&q=microservices

[40] Google Trends - REST API, Microservices a SOA[online]. In: . [cit. 2018-02-24]. Dostupné z: https://trends.google.cz/trends/explore?date=all&q=REST%20API,Microservices,SOA

[41] On the Move to Meaningful Internet Systems. OTM 2017 Workshops [online]. 2018 [cit. 2018-02-24]. ISBN 978-3-319-73805-5. Dostupné z: https://www.springer.com/us/book/9783319738048

[42] RODRÍGUEZ, Guillermo Horacio, Alvaro SORIA a Marcelo R. CAMPO. AI-based Web Service Composition: A Review [online]. 2015, , 1-8. [cit. 2018-02-25]. DOI: 10.1080/02564602.2015.1110061. Dostupné z: https://www.researchgate.net/publication/284766808_AI-based_Web_Service_Composition_A_Review

[43] Cloud orchestration (cloud orchestrator) [online]. December 2017 [cit. 2018-02-26]. Dostupné z: http://searchitoperations.techtarget.com/definition/cloud-orchestrator

[44] SUCHITRA VENUGOPAL. Cloud orchestration technologies: Explore your options [online]. 11 July 2016 [cit. 2018-02-26]. Dostupné z: https://www.ibm.com/developerworks/cloud/library/cl-cloud-orchestration-technologies-trs/index.html

[45] Google Trends - Cloud Orchestration [online]. In: . [cit. 2018-02-26]. Dostupné z: https://trends.google.cz/trends/explore?date=all&q=Cloud%20orchestration

[46] Automatická kompozice služeb [cit. 2018-04-10]. Dostupné z: https://git.cs.vsb.cz/kat0013/automatic-services-composition

[47] Connectors. Camunda Docs [online]. [cit. 2018-03-01]. Dostupné z: https://docs.camunda.org/manual/7.8/user-guide/process-engine/connectors/

[48] Spring Boot with Docker [online]. [cit. 2018-03-01]. Dostupné z: https://spring.io/guides/gs/spring-boot-docker/

[49] Spring Framework Overview: Version 5.0.5.RELEASE [online]. [cit. 2018-03-01]. Dostupné z: https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html

[50] Introducing Spring Boot [online]. [cit. 2018-03-01]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-introducing-spring-boot.html

[51] The Camunda BPM Manual [online]. [cit. 2018-03-10]. Dostupné z: https://docs.camunda.org/manual/7.8/

[52] What is Docker: OVERVIEW [online]. [cit. 2018-03-10]. Dostupné z: https://www.docker.com/what-docker

[53] Overview of Docker Compose [online]. [cit. 2018-03-15]. Dostupné z: https://docs.docker.com/compose/overview/

[54] What is MongoDB? [online]. [cit. 2018-03-15]. Dostupné z: https://www.mongodb.com/what-is-mongodb

[55] Thymeleaf [online]. [cit. 2018-03-15]. Dostupné z: https://www.thymeleaf.org/

[56] JUnit [online]. [cit. 2018-03-20]. Dostupné z: https://junit.org/junit4/faq.html

[57] Mockito [online]. [cit. 2018-03-20]. Dostupné z: http://site.mockito.org/

[58] Welcome to Apache Maven [online]. [cit. 2018-03-20]. Dostupné z: https://maven.apache.org/

[59] Maven Wrapper [online]. [cit. 2018-03-20]. Dostupné z: https://github.com/takari/maven-wrapper

A Seznam příloh na CD

- KAT0013.pdf ..... Diplomová práce ve formátu PDF/A
- automatic-services-composition ..... Git projekt prototypového řešení
  - asc-root ..... Základní Maven projekt
    - airplane-service ..... Maven projekt služby pro rezervaci letu
    - config-service ..... Maven projekt konfiguračního serveru
    - demo-service ..... Maven projekt pokusné služby
    - gateway-service ..... Maven projekt služby brány
    - hotel-service ..... Maven projekt služby pro rezervaci hotelu
    - orchestration-app ..... Maven projekt orchestrační aplikace
    - registry-service ..... Maven projekt registračního serveru
    - taxi-service ..... Maven projekt služby pro rezervaci taxi
    - docker-compose.dev.yml
    - docker-compose.yml
    - pom.xml
  - .gitignore
  - readme.md